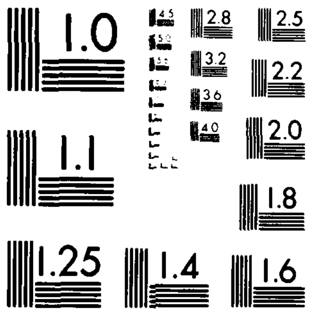
AD-A086 065 HARRIS CORP MELBOURNE FL GOVERNMENT COMMUNICATION SY--ETC F/G 9/2
MULTIPLE MICROPROCESSOR SYSTEM (MMS) DESIGN STUDY. VOLUME I.(U)
MAR 80 F30602-78-C-0114
UNCLASSIFIED RADC-TR-80-33-VOL-1 NL

1 of 2
AD
A086065


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

LEVEL #

RADC-TR-80-33, Vol I (of four)
Final Technical Report
March 1980

ADA086965

MULTIPLE MICROPROCESSOR SYSTEM (MMS) DESIGN STUDY

Harris Corporation

Government Communications Systems Division

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

80 6 30 022

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-33, Volume I (of four) has been reviewed and is approved for publication.

APPROVED: Michael A. Troutman

MICHAEL A. TROUTMAN, 1Lt, USAF
Project Engineer

APPROVED: [illegible]

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A	

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

19 **REPORT DOCUMENTATION PAGE** | READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 18 RADC-TR-80-33 - VOL-1 (of four)	2. GOVT ACCESSION NO. AD-A086065	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 MULTIPLE MICROPROCESSOR SYSTEM (MMS) DESIGN STUDY. Volume I.		5. TYPE OF REPORT & PERIOD COVERED 9 Final Technical Report.
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Harris Corporation Government Communications Systems Division		8. CONTRACT OR GRANT NUMBER(s) 15 F30602-78-C-0114
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harris Corporation Government Communications Systems Division P O Box 37, Melbourne FL 32901		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 16 25290104 17 01
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCA) Griffiss AFB NY 13441		12. REPORT DATE 11 March 1980
		13. NUMBER OF PAGES 150
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same 12 150		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: 1Lt Michael A. Troutman (ISCA)

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

multiple microprocessors	microprocessor
emulation	distributed architecture
computer architecture	performance measurement
architecture evaluation	total system design

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This is a design for a Multiple Microprocessor System (MMS) to be built as part of the System Architecture Evaluation Facility (SAEF) being developed at RADC. The MMS was designed to be used in the modeling of a wide range of multiprocessor configurations for the purpose of evaluating their suitability to unique Air Force data processing requirements. → to p. 1

411224 Gw

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

APPENDICES (Volume III)

EVALUATION

The work performed under this contract clearly demonstrates the feasibility of producing a general purpose, emulation-oriented tool for the purpose of modeling and evaluating multiprocessor computer architectures. The contractor through keen technological insight, has produced a high performance, user-oriented design for the Multiple Microprocessor System (MMS). This effort represents the successful initiation of one aspect of RADC's comprehensive investigation into the Total System Design (TSD) methodology aimed at providing more effective development tools for the system designer (TPO R3D, FY 78/79). The results of this contract will be used directly in a follow-on effort to actually produce a working MMS.

Michael A. Troutman

MICHAEL A. TROUTMAN, 1Lt, USAF
Project Engineer

INTRODUCTION

The conceptual design of the MMS, Multiple Microprocessor System, presented in this final MMS report is the result of a two year study based on a set of requirements specified by Rome Air Development Center. The final report does not attempt to present all of the initial requirements and constraints placed on the MMS or the design iterations the MMS has undergone. The final report does provide a detailed description of the choosen architecture of the MMS, the capabilities of the MMS, the components of the MMS, and how the components interact with each other.

In order for the reader to fully understand the final MMS report, he should possess an understanding of the driving force and the concepts behind the development of the MMS.

The MMS was conceived and designed to provide insight into the performance and operation of multiprocessor systems. Specifically it has been designed as a system design tool for real time multiprocessor systems. The MMS accomplishes this goal by providing:

1) a set of processing elements built for emulation.

(2) an extensive built-in performance monitor subsystem.

(3) a time align control mechanism to keep the emulators aligned in time.

A complete set of requirements that the MMS design meets is included in the"SAEF Interm Performance Specification" document dated October 1st, 1978. This report should be reviewed by the reader as background for this document.

The MMS final report is partioned into four volumes. These four volumes provide the conceptual design and specification of the MMS. Volume I is the conceptual hardware design of the MMS. Volume II is the conceptual software design of the MMS. Volume III contains the appendices referenced in Volumes I and II, except for the hardware and software specifications. Volume IV contains the MMS hardware and software specifications.

The hardware conceptual design is presented in Volume I in eight sections.

Section 1 covers an analysis of the MMS and target systems. The development of many of the concepts and requirements the MMS is based on are a result of this analysis.

Section 2 covers the system architecture of the MMS and covers such topics as: the busing structure, broadcast communications, and various interfaces used to interface the components of the MMS to the MMS bus.

Section 3 provides the conceptual hardware design of the 64 processing elements used in the MMS.

Section 4 describes how the time alignment mechanism of the MMS performs the function of maintaining the time dependency of an emulated process.

Section 5 covers the process of performing the emulation of the Input/output functions of a target system.

Section 6 discusses how to maintain control of emulated shared resources.

Section 7 describes how the performance monitoring mechanism of the MMS functions.

Section 8 covers the mechanical configuration and construction of the MMS.

Any appendix referenced in these eight sections of volume I can be found in volume III. The hardware specification of the MMS is found in volume IV.

The software conceptual design is contained in sections 10 through 23, which then forms Volume II.

Section 10 covers the functional description of the MMS hardware. This section identifies the needed software/firmware for support of the hardware.

Section 11 covers the needed software packages to interface with MMS hardware and firmware. This section forms the introduction to the MMS software.

Section 12 covers the requirements for diagnostic programs to test and verify the operational capacity of the hardware.

Section 13 identifies the need for an E-Code compiler.

Section 14 covers the microcode assembler to be used for writing and developing emulation packages.

Section 15 covers the executive program which is needed by all PE's of the MMS configuration.

Section 16 covers the System Generation Loader which interfaces with the user to configure the user TS hardware design as a set of emulation packages.

Section 17 covers the Run-time Executive which interfaces with the user to execute the defined TS.

Section 18 covers the Post-Mortem Dump and how the user can utilize this package to study the TS condition at a halt of execution.

Section 19 covers the PMS off-line Analysis package and how the user can use this package to analyze the performance data collected.

Section 20 covers the need and the scope of a Target System to be applied to MMS. This will verify the operation of MMS and will provide an example for future development on MMS.

Section 21 covers the User Scenario and the methodologies for interfacing with MMS during SGL and during RTE.

Section 22 covers the FCP computer and the trade-offs relative to several minicomputers.

Section 23 covers four general topics. The first topic is the identification of the MMS busing structure. The next, three topics are the following executives required by the units on the Master Segment:

- Executive for the I/O Processor
- Executive for the PMS Interface
- Executive for the Shared Resource Controller

1.0 MMS ANALYSIS

This section shall begin by giving a presentation of the goals of the analysis and an outline of the conclusions. The evolution of the analysis will be snapshot at five different stages. These stages will be presented in a historical sequence. Each stage will be presented in five sections: goals, inputs, method, results, and conclusions. The results and conclusions of the fifth stage are the most lengthy and involved and will be utilized as the results and conclusions of the analysis.

1.1 Goals

The first goal of the analysis was to provide an assurance of design feasibility and measure of the scope of design in an early stage in the project. In the early stages it was helpful to judge the design relative to cost and performance, and then to point out the direction in which the design should proceed. The MMS analysis aided in the specification of the major components in the system by relating sensitivity of system performance to each component in MMS. This provided an effective way to tradeoff system performance and cost. The analysis served as a sounding board to test ideas about the architecture of MMS and sometimes served to suggest architectural features that would increase the capabilities of MMS. The final goal was to demonstrate which target systems could realistically be emulated and estimate the respective emulation efficiency of those systems. The analysis was thus used as a tool to show the effectiveness of MMS on any particular target system.

1.2 Preview of Conclusions

The following topics represent a brief hypothesis of what was proven by this analysis:

1.2.1 Parallel memory bus.

1.2.2 Single system bus.

1.2.3 Organization of memory: distributed, local.

1.2.4 Necessity of distributed I/O.

1.2.5 Choice of microprogrammable Schottky logic.

1.2.6 EE range of 1.8, 3.7, 11.5, and 17 to 1 for 8, 16, 32, 64 bit processors, respectively. (EE = EMULATION EFFICIENCY)

1.2.7 Speed of 5 megatransfers/second for the MMS bus will be sufficient.

1.2.8 Data width on bus will be 16 or 32 bits.

1.2.9 Local memory will be 128K bytes addressable by byte or word.

1.2.10 ALU width will be 16 bits.

1.2.11 Shared resource controller will be microprogrammable Schottky logic.

1.3 Presentation of Five Model Phases

1.3.1 Worst Case Analysis

1.3.1.1 Goal

The goal of the worst case analysis is to generate an initial estimate of the gross information transfer rate.

1.3.1.2 Inputs

Four parameters are used as inputs to this stage of the analysis. These parameters and their worst case values are NP, the number of processors (64); BS, the bit size of the processor

(64); IMA, the number of memory accesses per instruction for a 64 bit processor (1.36); and NIPS, the number of instructions per second for a 64 bit processor (1.5 million).

1.3.1.3 Method

The method of calculation for the transfer rate is the multiplication of the input parameters. NP X BS X IMA X NIPS yields 8.4 bits per second transferred for memory accesses alone for a serial busing structure.

1.3.1.4 Results

The results show that for raw memory transfers alone, not counting any overhead associated with message format, a rate of 8.3 GHz is needed for a one-to-one emulation efficiency. This rate is not feasible with existing technology. Even at the low point of the design criteria of 100:1 emulation slowdown, the rate is 83 MHz.

1.3.1.5 Conclusions

The primary conclusion drawn from these results is that a serial memory bus is out of the question because of the speed. Therefore a parallel bus was chosen.

1.3.2 Target System Analysis

1.3.2.1 Goals

In order to specify the MMS system it is necessary to understand the size and variety of target systems that will be emulated.

It is also necessary to analyze the communication structure and rates in order to parameterize the target system. It is impossible to directly translate the large variety of existing target system hardware to a fixed set of MMS hardware. However, each target system's physical transfers can be calssified and summed into a small group of functional transfers. These functional transfers can be distributed in the physical MMS network through a translation or mapping of the aggregate rates. It is for this reason that these average aggregate rates are valuable.

1.3.2.2 Inputs

The inputs to the target system analysis are a set of statistical parameters. These parameters and their estimated values are listed in Appendix B and Appendix C, respectively.

1.3.2.3 Method

The examination of a multiple processor network was performed on a processor by processor basis. Each processor was analyzed in detail by an examination of instructions. Through the study of instruction mixes, particularly the DIS MIX, an estimation was made of the number of times a memory is accessed for the average instruction. Using the DIS MIX* applied to the currently popular microprocessors and computers available, it was decided that an average number would be valid if the determination were made separately for each different machine bit size. Among machines of the same bit size the quantity of memory accesses was judged to fall in the same range despite the fact that instructions and instruction formats could be vastly different. The memory access parameter under discussion was called the Index of Memory Accesses (IMA).

* Digital integrating subsystem mix for military uses of computers.

It was computed through the use of the table below.

Type	Examples	Weight	Machine Bit Size			
			8	16	32	64
Operand Fetch	Load, Store	36%	4	3	2	2
Long Instr.	Branch	30%	3	2	1	1
Short Instr.	Reg. Operations	34%	1	1	1	1

The results are tabulated below:

IMA (8) = 2.68

IMA (16) = 2.02

IMA (32) = 1.36

IMA (64) = 1.36

A number of other parameters which describe a processor in a target system were chosen according to the frequency of their activity relative to the number of instructions executed. These parameters were used to provide a conceptually easy way to view the performance of a processor. One of the goals of the analysis was to obtain target system aggregate rates for various functional transfers. Parameters that describe the frequency of input/output (I/O), interprocessor communications, and shared resource requests were very helpful in achieving that goal. A complete list of the parameters and their definitions, as well as the values chosen for them, are presented in Appendix B and C.

Four Functional Rates - There are four logical functions that exist in target systems which require data and/or control transfers between elements in a multiprocessor system. These functions are memory, input/output, interprocessor communications, and shared resource control logic. The aggregate rates of these functions are determined for eventual translation to a given MMS architecture. These aggregate rates are defined in terms of the number of transfers per second of execution time. The transfer width is not specified here. Anytime data is passed, regardless of the number of bits that are passed in parallel in the target system, one transfer is summed to the total.

Fundamental Calculation - The model is constructed so that one fundamental calculation is necessary to obtain each logical rate. Parameters were chosen that represent the frequency of a function as measured relative to the instruction. These parameters are called indices and are specified by an "I" preceding the rest of the parameter identification. The acronyms for these parameters are IMA, IIO, IIPC, and ISRR (See Appendix B for definitions). Each parameter represents an occurrence of the function per instruction executed. Another parameter is NIPS, the average number of instructions executed per second. NIPS is otherwise known as the KOP (thousands of operations per second) rating of the processor determined by an instruction mix. The aggregate rate for a function is obtained by multiplying NIPS and each index summed over all the processors in the system. The equations for determining aggregate rates are listed in Appendix D.

The values of each parameter were specified over a statistical range. Sample target systems were generated by randomly selecting a value for each parameter from the distribtuion specified for that parameter. Thousands of target systems were sampled and a histogram of each of the four functional rates was computed.

From each histogram, three points were marked to present a concise picture of the results. The first point, "A", represented the transfer rate at which the probability of finding a target system with fewer transfers was 0.1. Point "B" represented the median transfer rate. Lastly, point "C" represented the transfer rate for which the probability of finding a target system with fewer transfers was 0.9.

1.3.2.4 Results

A sample histogram is shown in Figure 1.3.2.4. The points marked "A", "B", and "C" represent the 10%, 50%, and 90% values discussed in the previous section.

The "A", "B", and "C" transfers rates computed for each functional rate are given in Table 1.3.2.4.

1.3.2.5 Conclusions

The large transfer rates associated with memory access confirm the need for a wide parallel bus structure. Theoretically, the rate represents the number of memory transfers which must be supported by the MMS bus structure. This assumes a bus structure sufficiently wide enough to handle the largest number of bits that the user will require sent in a single logical transfer.

The reciprocal of the memory transfer rate shown that approximately 30ns is available for the bus to send a request to a memory, cycle the memory, and send a reply to the requesting device. This is clearly beyond the scope of the technology available. Two design possibilities for reducing the bandwidth of the MMS busing structure are suggested here.

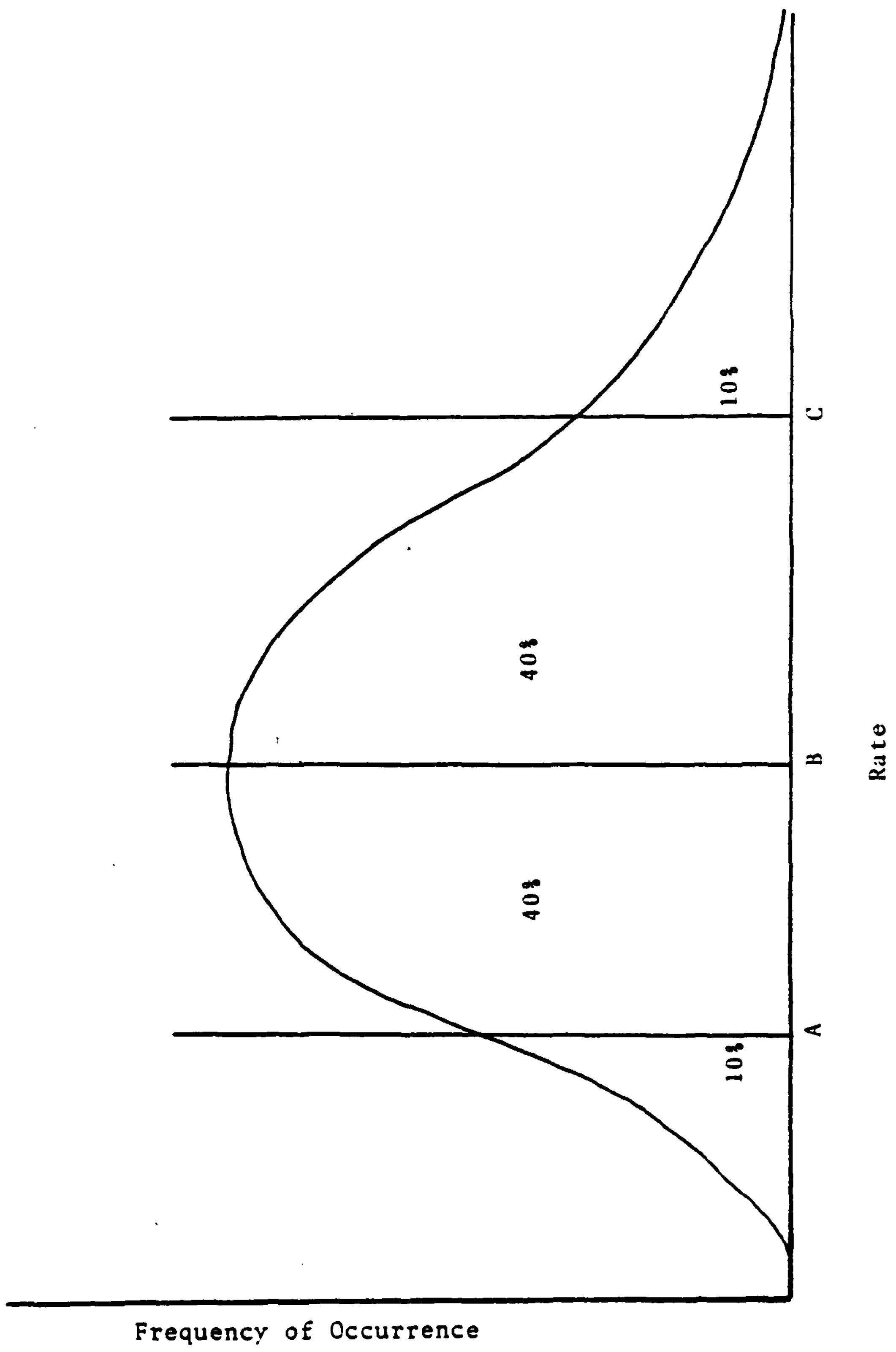


Figure 1.3.2.4

TARGET SYSTEM RESULTS

	A	B	C
Memory Transfer Rate (MTR)	[illegible]	[illegible].4M	41.7M
Shared Resource Request Rate (SRRR)	77.7K	320.4K	679.6K
Input/Output Transfer Rate (IOTR)	130.2K	269.0K	468.8K
Interprocessor Message Rate (IMR)	12.1K	24.9K	45.8K

(Units in transfers/second)

Table 1.3.2.4

First, the use of local memory within the PE removes the bus from the memory access path for a large number of the memory requests. Second, the establishment of these local memories as separate memory modules which may be accessed from the memory bus as separately functioning intelligent modules allows the bus to operate independently of the memory cycle time. Thus, the bus cycle time is decreased.

A striking result of the target system analysis is the large difference in the rate calculated for memory transfers and the rates calculated for I/O, interprocessor communications, and shared resource requests. The difference in transfer rates indicates two possibilities for implementing the MMS functional busses. The first possibility is the use of relatively low bandwidth parallel busses for the I/O and SRR communications, and a serial bus for IPC's. The second possibility is to perform all communications over a single bus, for which the total transfer rate would be approximately equal to the memory transfer rate.

Performance may suffer when separate physical busses are used to implement the aggregate rates associated with each respective function. There will be times when the executing code is I/O intensive. In this case the I/O bus will overload and retard total system performance. A single bus helps to alleviate this problem. With respect to time, the peak use of any one function will correspond to the sparse use of the remainder of the functions. The deviations of the functional rates from their aggregate averages will tend to cancel one another. By using a unified bus, no loss of performance should be incurred.

A second reason for possible performance degradation involves the misleading nature of the aggregate average method. For example, a separate bus implemented at the SRC aggregate rate would not be sufficient to meet the speed requirements for system synchronization. In the target system, shared resource arbitration occurs at a very fast speed. However, the frequency of arbitration may not occur as fast as the actual arbitration process. The aggregate rate represents the frequency. For complete synchronization in MMS, the arbitration speed implemented should approximate the arbitration speed of the target system action. The specialized SRC bus would not be fast enough. It could be argued that the specialized SRC bus could be implemented at a faster speed, but this speed would equal or better the speed proposed for the unified bus. Since this unified bus better approximates the speed desired and represents a better solution than four separate busses, it is recommended.

Except in the case for the interprocessor message rate, variations in the ten and ninety percent values did not indicate a order of magnitude change in the transfer rates. This indicates two major points. First, there is sufficient uniformity in target systems to present some assurance that one system of hardware can be built to emulate a variety of target systems. Second is the realization that not all target systems which could be emulated were generated by this statistical analysis.

1.3.3 Initial MMS Model

1.3.3.1 Goal

The goal of the initial MMS model was to determine some initial estimates for MMS communications rates.

1.3.3.2 Inputs

The initial MMS model was developed by modifying the target system model. The modified target system model was used to determine the MMS transfer rates needed to perform a one to one performance emulation of a target system.

Basically there are two new parameters input to the target system analysis. The first parameter is the local memory size (LMS). The LMS parameter is used to specify the total number of memory bits associated with each PE in MMS. The second parameter is the width of the bus (BWD). The BWD parameter is used to specify the width of the data portion of the MMS busing structure.

These two parameters, along with the parameters which specify the bit size (BS) and the memory space (MS) associated with each processor in the target system, are used to calculate a local memory factor (LMF) and a weighting factor (NF). These two factors are used to transform a target system memory transfer rate into an MMS memory transfer rate.

When a PE directly accesses its own local memory, the functional MMS memory bus does not have to be accessed. LMF is used to factor the effects of a local memory being accessed directly by a PE into the memory transfer rate (MTR) calculation. LMF is calculated in the following manner:

$$\text{LMF} = \frac{(\text{MS})\ (\text{BS}) - \text{LMS}}{(\text{MS})\ (\text{BS})}$$

When calculating, LMF is set equal to one if $(\text{MS}) \times (\text{BS}) < \text{LMS}$.

The assumptions made when defining LMF are:

1. A PE will always execute instructions from its local memory unless the memory space required for storing macrocode is greater than the size of a single local memory.
2. An equal probability exists that a PE will access any part of its designated MMS memory space.

NF is used to factor the effects of a fixed MMS bus width into the MTR calculation. NF is calculated in the following manner:

$$NF = \frac{BS}{BWD}$$

When calculating, NF is set equal to one if $BS < BWD$.

The assumptions made when defining NF are:

1. The entire bandwidth of the MMS bus will not be utilized when emulating processors for which $BS < BWD$.
2. Emulating an N-bit processor will require N total bits to be transferred N/BWD times.

The equation to calculate the MMS MTR follows.

1.3.3.3 Method

$$MTR = \sum_{i=1}^{NP} IMA_i \times NIPS_i \times LMF_i \times NF_i$$

1.3.3.4 Results

Four different configurations of MMS were charted with the same statistical variations done for each target system. Shared resource controller rates, input/output transfer rates, and interprocessor message rates were not affected by these variations and were not included in this part of the analysis. The new memory transfer rates are tabulated below. Note: The results follow the "ABC" format of the histogram previously discussed.

LMS	BWD	A	B	C
* 0	∞	17.8 M	29.4 M	41.7 M
16K X 16	16	13.3 M	24.4 M	37.8 M
16K X 32	32	6.94M	13.1 M	20.0 M
64K X 16	16	1.94M	6.11M	13.1 M
64K X 32	32	46.3 K	46.3 K	2.69M

*Target System Rates

1.3.3.5 Conclusions

One conclusion to be emphasized is that local memory does indeed drastically reduce the rate at which the memory bus must be operated to perform an emulation. The important point is that the MMS transfer rate can be reduced to a value which doesn't exceed the limits of high speed Schottky logic.

Secondly, the use of small blocks of memory which may be accessed in parallel, whether they be situated local to the PE or not, is essential to obtain high bus throughput. With the creation of a memory block that has the ability to receive requests, make the access, and request the bus for transmission, the memory cycle time may be separated from the bus cycle time.

By including a memory block with each PE, it reduces the number of users trying to access that block. With the memory placed on the bus in one block, the wait time for a PE to use the single memory block would be many times greater than the access of the physical memory.

With the aid of the final MMS analysis program, the access time to the MMS bus and the emulation efficiency of a target system consisting of 16 eight-bit processors was computed. For a bus structure which utilized only one block of memory with its access time included in the bus cycle time, the latency time to the memory was 14.6μs with a system EE of 15.59 to 1. With 64K local memory blocks, the latency time was 1.67ns with EE of 1.79 to 1.

It was at this point in the analysis that a decision was made to begin concentrating on parallel bus architectures that could yield a speed of approximately ten mega-transfers per second. This ballpark figure was determined by viewing the fifty percent value for memory transfer rates with a 64K X 16 local memory size.

1.3.4 Inclusion of Emulation Efficiency and Component Loadings

1.3.4.1 Goals

In order to achieve an improved estimate of the specification of components in the system, it is necessary to get a better handle on how fast the PE's can operate relative to real time. This is called emulation efficiency (EE), and is given by the ratio of the MMS emulation time to pseudo-time (time elapsed for a real time target system run performed on the target system). A decrease in the emulation efficiency from the previously set 1:1 to a specifically calculated efficiency could show a decrease in the demands placed on MMS components.

Also needed in the analysis is a way to determine which components in the system are acting as bottlenecks to the rest of the system. The goal is to alleviate such bottlenecks and make a balance on performance demands placed on all components of the system.

1.3.4.2 Inputs

The major input to this phase of the analysis is the inclusion of the concept of the bus and its protocol. The MMS busing structure used is a synchronous bus with the memory cycle time separated from the bus cycle time. Information from one device is sent to another device in one bus cycle. If a reply from the device is required, the "think time" of that device would be concurrent with other bus cycles. When the reply is prepared, that device would bid for the bus and perform the transfer on the bus in one bus cycle like any other device or processor. This means that for every memory read operation on the bus, there is a requirement of two bus cycles as well as the two latency times incurred while bidding for the bus. These two operations apply to shared resource request/grant actions with the shared resource controller, as well as to any other action in the system where two way communication is necessary. It is also assumed, for the purpose of this analysis, that ninety percent of all memory accesses are read accesses. This assumption is based on numerous cache memory studies which have been made.

For the purpose of determining the capabilities and loading on four major components in the system, maximum rates and additional parameters were devised. Maximum rates of ten mega-transfers per second for the bus, 3 hundred thousand request/grant operations per second for the shared resource controller, and ten thousand input/output operations per second for the I/O processor were assumed. Input/output operations were broken down and parameterized into three basic kinds. First there were certain types of simple I/O that could be handled entirely within the PE such that no action was required by the I/O processor at all. An index per I/O instruction was assigned for this type of I/O called IIOC, the index of the input/output controller function situated within the PE. IIOC was estimated at this time to have a value of 0.2. The second type of I/O was considered to be block transfers from a disk or a tape to the local memory of the PE or to some memory module on the bus. This would require DMA action by the I/O processor but no actual processing time. The actual handling of the I/O emulation would again take place inside the PE with the I/O controller function. The parameter chosen to represent this type of I/O was IMMIO, the index of memory mapped input/output. It was estimated to have a value of approximately 0.5. The third type of I/O was considered to be environmental in nature and would require the attention of the I/O processor. These types of I/O operations were viewed to be the heavy load on the I/O processor. Additionally, any I/O which was considered shared as defined by the IOWF, I/O weighting factor, was considered subject to action by the I/O processor. IOWF was given a value of 0.95.

The last component in the system whose maximum rate was needed was the PE itself. Estimates were made for the number of instructions per second that a processing element could emulate relative to the size of the target system processor and the ALU width present in the PE. The following table shows these estimates in terms of thousands of instructions per second.

MMS ALU Width	Target System Bit Size 8	16	32	64	
16	300	160	120	100	PE KOP
32	300	200	200	160	Rating

1.3.4.3 Method

The four functional rates of I/O, IPC, SRR, and memory are translated into three physical rates representing the MMS bus, the shared resource controller, and the I/O processor. These three physical rates are calculated as follows:

$$BTR = MTR \times 1.9 + SRRR \times 2\ IMR + IOTR$$

$$\text{Rate for SRC} = SRRR$$

$$IOPR = \sum_{i=1}^{NP} IIO_i \times (1 - IOWF_i \times (IIOC_i + IMMIO_i)) \times NIPS_i$$

The bus transfer rate (BTR) involves a combination of all of the functional rates adjusted by a weighting factor which represents the number of bus cycles necessary for a complete transaction. The factor of 1.9 represents a combination of the two bus cycles needed for memory read and the frequency of read versus write operations. The shared resource controller rate is identical to the shared resource request rate previously discussed. The IOPR is the old input/output transfer rate modified to include only those I/O transfers that involve computing action by the I/O processor.

By calculating the BTR, SRRR, IOPR, and PE emulation efficiency (EE), an overall MMS emulation efficiency can be determined in three steps.

The first step is to determine the EE for each MMS PE.

EE is determined for each MMS PE as the ratio of the instructions execution rate of the emulated processor (NIPS) to the estimated MMS PE KOP rate. The EE ratio calculated indicates how much slower an MMS PE is compared to a target system processor in terms of instruction execution.

The second step is to determine the PE with the maximum slowdown ratio. (The slowdown ratio is the inverse of the emulation efficiency.) All of the results of emulation efficiency are presented in this slowdown form. This ratio is used as the first pass emulation efficiency of the system.

The third step is to determine the loading on the bus, SRC, and the IOP. Each loading is computed as a ratio of the rate calculated (BTR, STCR, IOPR) to the maximum rates assigned for the component given in Section 1.3.4.2. If, upon examination of these loadings, it is determined that all loadings are less than or equal to one hundred percent, then the emulation efficiency and component loadings calculated are correct. Otherwise, the emulation slowdown is multiplied by the maximum loading while each of the loadings are divided by this maximum loading. Note: For this case, one component is always shown as having a 100% loading.

Conversely, if neither the bus, IOP, or SRC is found to be the bottleneck in the system, it follows that the true bottleneck in the system is the processing element. Note the choice of the worst case slowdown of a PE as the standard. The use of worst loading as a division factor is possible due to the forced synchronization placed on the system by pseudo-time.

1.3.4.4 Results

The results for this phase of the analysis are listed on the following pages. Emulation slowdown ratio, bus loading, SRC loading, and IOP loading are shown in Table 1.3.4.4, listing the "A", "B", and "C" values from the respective histogram. The numbers beside the asterisks represent the number of times that particular component was the bottleneck in the system.

Figure 1.3.4.4 shows a histogram of PE utilization. This histogram represents the utilization efficiency of all the PE's in the target systems generated. This utilization efficiency is actually the percentage of time that the PE is executing instructions and/or waiting on accesses versus the total run time (including the time stopped by pseudo-time for synchronization). The numbers eight, sixteen, and thirty-two marked on the graph represent the regions where eight, sixteen, and thirty-two bit target system processors are expected to fall.

1.3.4.5 Conclusions

This phase of the analysis revealed the I/O processor to be the bottleneck for a large number of target systems. In roughly 20% of the target systems run, the load on the IOP was the factor which degraded the emulation efficiency the most.

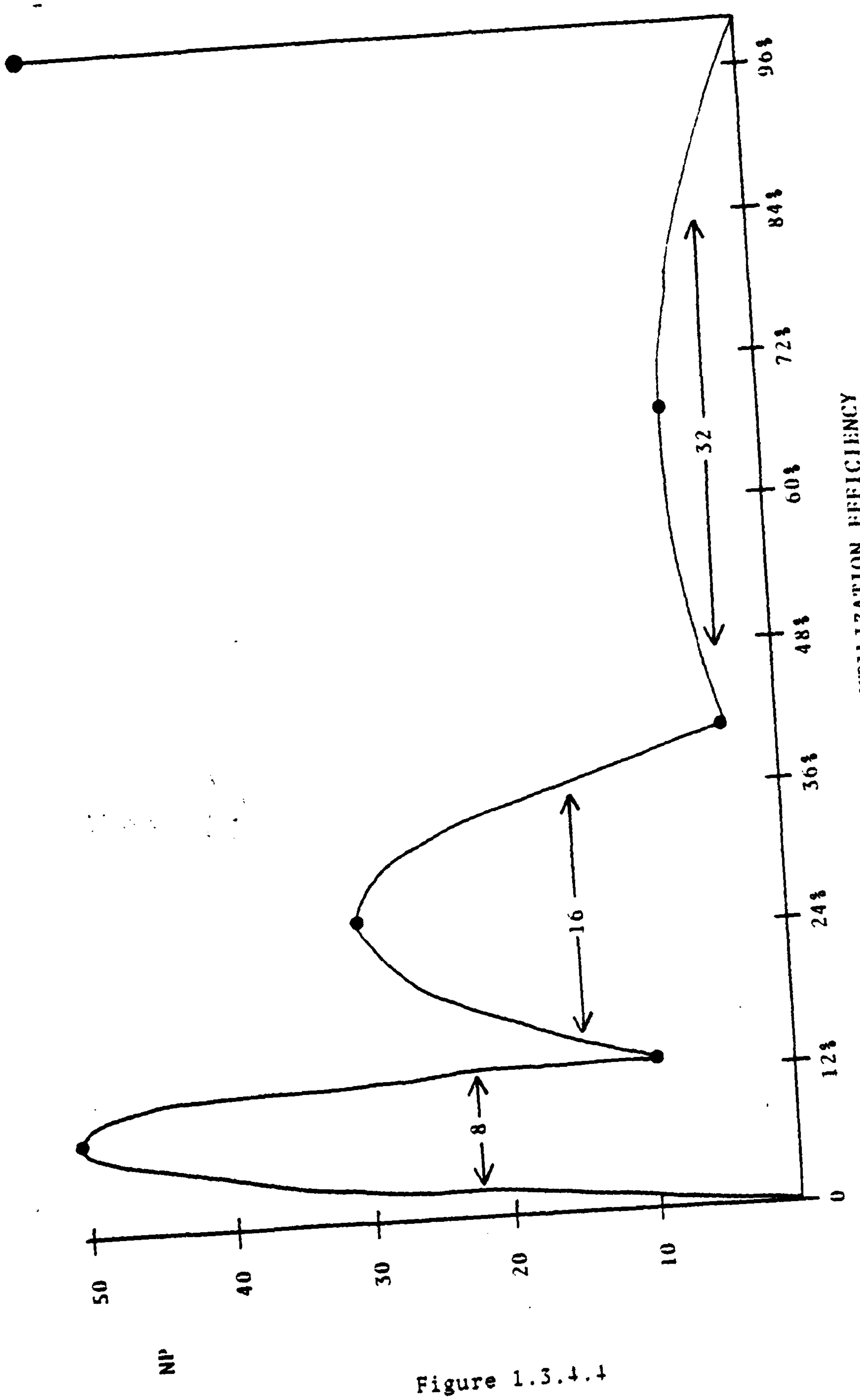
The rate input to this phase for the IOP is ten thousand IOP actions per second. This rate represents the upper limit on the rate at which a computer can be interrupted. Therefore, to keep efficiency at a reasonable level, IOP actions must be reduced drastically. One method to reduce IOP actions is to concentrate heavily on distributing I/O emulation throughout the PE's as much

EE AND COMPONENT LOADINGS

LMS		EE	BUS LD	SRC LD	IOP LD
16K X 16	A	19.23	.458	.025	.317
	B	14.5	.750	.083	.567
	C	10.0	1.000*$_{11}$	.158	1.000*$_{6}$
16K X 32	A	16.1	.267	.017	.283
	B	13.3	.400	.075	.625
	C	7.6	.508	.192	1.000*$_{8}$
64K X 16	A	17.54	.067	.017	.300
	B	14.92	.192	.100	.533
	C	10.75	.325	.158	1.000*$_{8}$
64K X 32	A	18.87	.005	.025	.292
	B	14.5	.025	.067	.625
	C	11.63	.058	.167	1.000*$_{10}$

*These numbers show how often this component was the bottleneck in the fifty systems generated and analyzed.

Table 1.3.4.4



HISTOGRAM OF PE UTILIZATION EFFICIENCY
(Statistical Target System)

Figure 1.3.4.4

as possible.

Ancther method to reduce IOP actions is concerned with environmental data calculations. Environmental type calculations can be considered as I/O simulation at best. It can not be expected that these external simulations of environment will be able to keep up with the faster emulation done on MMS. Therefore, the timing specifications for the purpose of analysis will assume that no more than one percent of I/O will be of the environment calculation type. Of course, the user maintains the viable alternative to attempt to emulate his environment in an unused PE. In this case, the performance of the system would not be so greatly affected.

This phase of the analysis also reveals the bus to be a bottleneck when each PE contains only a small amount of local memory. However, with a suitable amount of local memory, the bus is seen to be no more than 30 to 40 percent loaded. Therefore, the need for a bus running a 10MHz is questionable, 5MHz seems adequate to handle the load with a suitable local memory size.

A final point brought out by this analysis is that a 32-bit or 64-bit processor is the rate determining processor as shown by the PE utilization histogram. Again, this is due to the forced synchronization and the facts that large target machines are built to run faster than smaller machines, and are more difficult to emulate than smaller machines. This brings up a problem as to what design emphasis should be taken in the MMS. Ideally geared toward micro machines of which the most common of these being 8-bit and 16-bit machines, the performance is obviously directly reflective of the largest bit size processor in the target system. If a 32-bit processor exsits in a target system, system performance is not affected at all by optimizing the design for a 16-bit processor. Even if there are ten times the number of 16-bit processors present than there are 32-bit processors, the 32-bit processors are still the determining factor for emulation efficiency. On the other hand, if the target system contains only eight and sixteen-bit processors, performance is not improved in the slightest by designing toward 32-bit machine emulation.

A more complete and fulfilled model is needed. A better estimation of internal PE instruction timing based on specifications placed on components is needed, and bus analysis should include access time. A study of specific target systems, both homogeneous and heterogeneous, is needed.

1.3.5 Final MMS Model

1.3.5.1 Goals

The goal of the final MMS model is to obtain a complete and accurate picture of MMS. In order to accomplish this, the instruction cycle time must be dissected into two major categories. The first category consists of those basic functions that must be handled inside the instruction execution unit. The second category consists of the parts of the instruction execution that require external access. The attempt should be made to characterize the internal actions in terms of micro cycles and the external actions in terms of time paths to different devices. Specifically, for external actions the latency time to the MMS bus must be defined. This latency time is used to determine accesses and other uses of the bus. These time paths can be fed back into the emulation instruction cycle time (ICT), which allows a more accurate measure of the KOP rates of the PE's.

In order to obtain a complete MMS model, a wider variety of specific target systems need to be input for analysis. Also, individual component values at the processor level, or even at the instruction level, need to be input so that sensitivity studies may be done for any component with respect to the system's performance and cost.

The last goal will be to present, in an orderly manner, the capabilities of MMS. The analysis will also present emulation efficiency and component loadings of MMS when specific target systems are analyzed.

1.3.5.2 Inputs

New inputs to this phase of the analysis included specifications on various components within the PE, as well as timing specifications of the different functions and transfers performed internally and externally to the PE. Specifically, the following MMS parameters were defined; the cycle time of the memory (MCT), the micro cycle time (μCT), the width of the ALU, (ALUWD), service time of the MMS bus (S), service time of the SRC (SRCSV), service time of the IOP (IOPSV), and average translation time for the memory interface (MI).

A list of all MMS parameters is found in Appendix E. A list representing some of the MMS configurations which were input to this phase is shown in Appendix F. Make particular note of configurations F and G. These are the two most favored configurations.

Another major input was the functional analysis of the number of micro cycles necessary to implement a macro instruction. See Table 1.3.5.2. Taking into account only those functions of emulation that are performed internally to the instruction execution unit, a breakdown of the number of micro cycles per function was charted according to target system bit size and MMS ALU width.

Instruction execution is defined by five functions. The five functions are instruction decode, address calculations, program counter increments, arithmetic, and I/O and interrupt handling. The sum of the micro cycles for these functions is called ESIMP. This parameter represents the simple execution functions.

Note that decode time is the largest factor in instruction emulation. This is disproportionate to the decode time in the target machine. The reasons for the disproportionment are threefold. First, the MMS decode is being done in firmware rather than hardware. Secondly, MMS flexibility requirements prohibit fast emulation of arbitrary instruction formats. Thirdly, the actual decoding of the instruction may at times be overlapped with execution in wide formats.

The remaining emulation time deals with the time spent performing accesses to the other devices not internal to the IEU. The path for each type of access is traced through MMS. Each path represents the accumulation of the average times spent in each component. Once these path times are calibrated, each is weighted by the appropriate index that corresponds to the frequency of the action's occurrence relative to the instruction.

Following is a list of external paths and the equations used to calculate external action times.

PATH	INDEX	TIME
Bus Memory Write	LMF X IMA X 0.1 X NF	MI/NF + A + 2S
Local Memory Write	(I-LMF) X IMA X 0.1 X NF	MI/NF
Bus Memory Read	LMF X IMA X 0.9 X NF	NI + NCT + 2A
Local Memory Read	(I-LMF) X IMA X 0.9 X NF	MI/NF + MCT
Shared Resource Req.	ISRR	SRCSV + 2A 2S
IOP Read Request	IIOP X IIO X 0.9 X NF	IOPSV + MI/NF + 2A + 2S
IOP Write Request	IIOP X IIO X 0.9 X NF	IOPSV + MI/NF + A + 2S
MEM Mapped Write	IMMIO X IIO X 0.1 X NF	MI/NP + A + 2S
MEM Mapped Read	IMMIO X IIO X 0.9 X NF	MI/NF + MCT + 2A + 2S
IPC (All Writes)	IIPC	A + 2S
DMA From FCP Disk	IIO X IMMIO X NF	MCT + S

INTERNAL FUNCTION MICRO CYCLES

FUNCTION	ALUWD = 8			
	BS = 8	16	32	64
Decode	4	8	10	10
Arithmetic	1.5	3	6	12
ADDR Calc	5.36	7.07	5.44	5.44
INCR PC	2	3.5	4	4
I/O & INT	2	2	2	2
ESIMP (SUM)	14.86	23.57	27.44	33.44

FUNCTION	ALUWD = 16			
	BS = 8	16	32	64
Decode	4	8	10	10
	1.5	1.5	3	6
ADDR Calc	2.68	3.535	2.72	2.72
INCR PC	1	1.75	2	2
I/O & INT	2	2	2	2
ESIMP (SUM)	11.18	16.789	19.72	22.12

FUNCTION	ALUWD = 32			
	BS = 8	16	32	64
Decode	4	8	10	10
Arithmetic	1.5	1.5	1.5	3
ADDR Calc	2.68	2.02	1.36	1.36
INCR PC	1	1	1	1
I/O & INT	2	2	2	2
ESIMP (SUM)	11.18	14.52	15.68	17.86

Table 1.3.5.2

By summing the external action times and the internal action times, an average instruction execution time can be calculated. The instruction execution time is calculated as follows:

$$ICT = ESIMP \times MICRO\ CYCLE\ TIME + \sum_{i=1}^{NP} INDEX_i \times TIME_i$$

1.3.5.3 Method

The method of calculating emulation efficiency is as follows. An MMS configuration is input in its entirety by inputting all MMS parameters. The target system is then input one processor at a time by specifying all indices and parameters associated with each processor. The number of accesses to the MMS bus per instruction executed is calculated for each processor as follows:

$$NAPI = (LMF \times NF \times 1.9 \times IMA) + (2 \times ISRR) + (IIPC) + (NF \times (IIOP + 2 \times IMMIO) \times IIO).$$

The ICT and EE for each processor must now be determined. A judgement to determine ICT and EE is made when each PE is defined and input to the analysis to determine which processor has the worst EE input.. The NAPI for all processors is normalized to NAPI of the processor with the worst EE to account for forced synchronization in MMS. A running sum is kept of NAPI. If the input processor does have the worst EE, then the running sum of NAPI is normalized to the input processor and its NAPI is summed. The calculated ICT time is the base value which does not include bus access time. When all processors have been input, the average NAPI is calculated; then the average execution time between bus accesses, E, is calculated. The total ICT is calculated as follows:

$$ICT_{TOT} = ICT_{INT} + NAPI \times A$$

The system EE is determined by the following equation:

$$EE = ICT_{TOT} \times (\text{NIPS or Worst Processor EE}).$$

A flow graph of the ICT and EE calculations is given in Figure 1.3.5.3.

The method of calculating access time to a device that is shared by many processors is difficult to define. The access time depends on the number of processors using the device, the frequency at which the device is requested by each, and the amount of access time that the device requires while locking out other processors from its use. The frequency of using the MMS bus is entirely dependent on the amount of access time to the bus, while the access time to the bus is dependent on the frequency that all processors are requesting the bus.

The problem is a classic finite population queueing system. The solution of the general case is presented in Introduction to Computer Architecture. The following formulas are derived:

$$W = \frac{n}{\mu\ (1 - po)} - \frac{1}{\lambda}$$

$$\text{where } po = \left\{ \sum_{0 \le i \le N} \frac{n_{i}}{(n - i)} p^{i} \right\}^{-1}$$

$$\text{and } p = \frac{\lambda}{\mu}$$

These formulas were applied to the MMS system in the following manner:

$$W = S + A$$

$$\lambda = 1/E$$

$$\mu = 1/S$$

$$n = NP$$

$$E = \frac{ICT}{NAPI}$$

The following definitions apply:

S = Service time on MMS bus

A = Access time to or latency time of MMS bus

NP = The number of processors in the system

ICT = The average instruction time of an instruction executed in a PE.

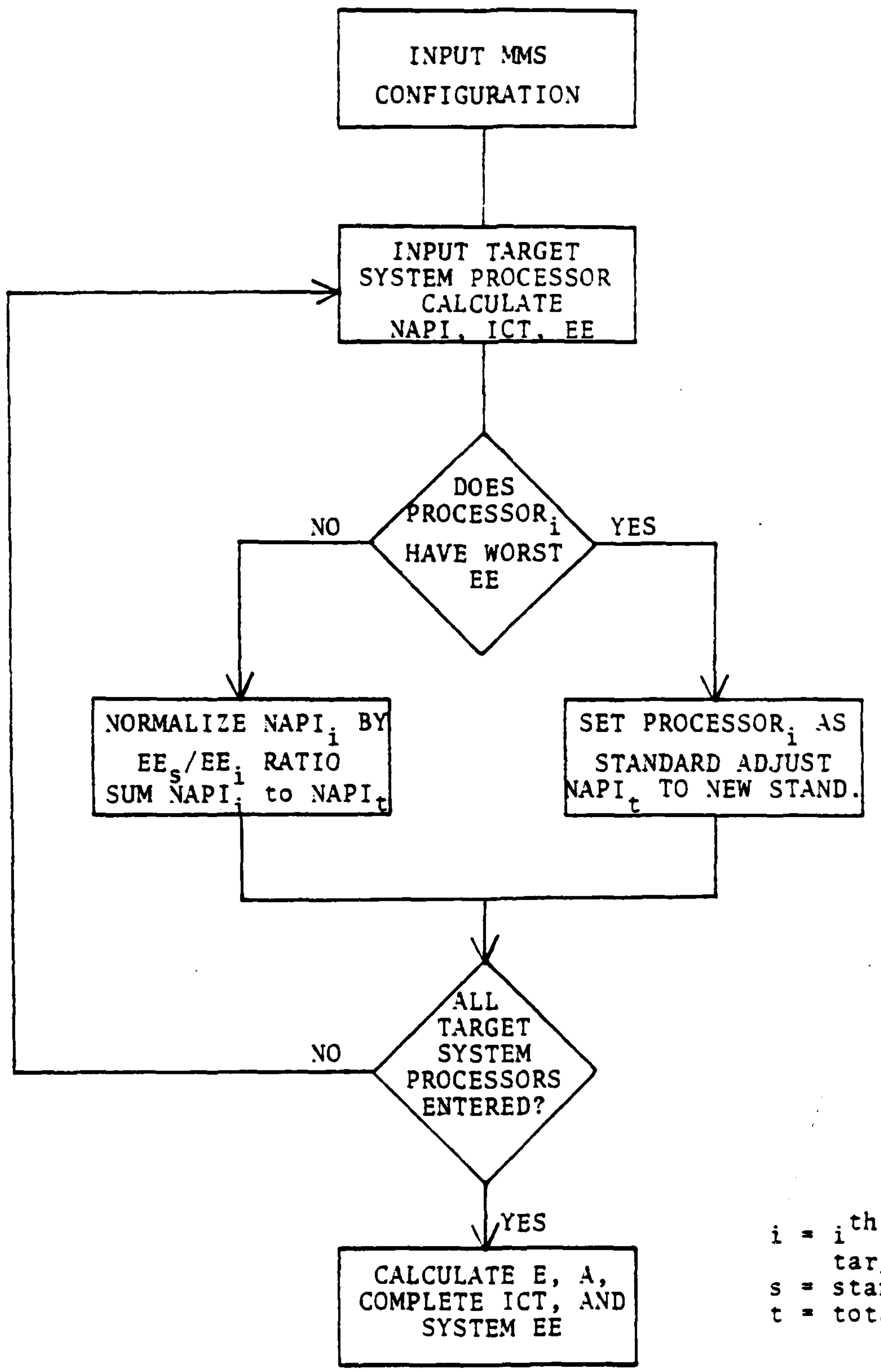
E = The average amount of time that a PE executes on an instruction before a request needs to be made to the MMS bus. Or the inverse of the rate at which requests are made to the MMS due to the average processor.

NAPI = The average number of accesses which are made to the MMS bus per PE.

Substituting these MMS parameters into the equations above gives the following:

$$A + S = \frac{NP}{\left(\frac{1}{S}\right)\left(1 - \left[\sum_{i=0}^{NP} \frac{NP\,!}{(NP - i)\,!}\left(\frac{S}{E}\right)^{i}\right]^{-1}\right)} - E$$

$$A = \frac{(NP)\ X\ (S)}{1 - \frac{1}{\left[\sum_{i=0}^{NP}\left(\frac{NP\,!}{(NP - i)\,!}\right)\left(\frac{S}{E}\right)^{i}\right]}} - E - S$$


INPUT MMS CONFIGURATION
INPUT TARGET SYSTEM PROCESSOR CALCULATE NAPI, ICT, EE
DOES PROCESSOR$_i$ HAVE WORST EE
NO
YES
NORMALIZE NAPI$_i$ BY EE$_s$/EE$_i$ RATIO SUM NAPI$_i$ to NAPI$_t$
SET PROCESSOR$_i$ AS STANDARD ADJUST NAPI$_t$ TO NEW STAND.
ALL TARGET SYSTEM PROCESSORS ENTERED?
NO
YES
CALCULATE E, A, COMPLETE ICT, AND SYSTEM EE

i = i^th processor in target system
s = standard
t = total

FINAL MMS MODEL METHOD

Figure 1.3.5.3

1.3.5.4 Results

The results for the fifth stage of analysis are essentially the results of the analysis as a whole. They are presented in a series of ten charts or graphs which show the emulation efficiency, bus speed, bus latency time, and shared resource controller loading for a wide span of target systems. Also presented here is a graphic representative of decisions made about memory organization and system bus width relative to system cost performance. Most of the last phase of analysis is done assuming the architecture of MMS to be divided into sixteen processor segments (see Section 2.5 for more details). The basic layout for many of the graphs shows the target system segment as the abcissa.

No analysis has been done on intersegment communication frequency or its effect on emulation efficiency. However, it was assumed if there is no more than ten percent frequency of inter-segment transfers, then there will be no more than twenty-five percent drop in overall emulation efficiency. Should one or more processing elements have ninety percent or more of its transfers cross segment boundaries, emulation efficiency would drop significantly.

Along the abcissa on many of the graphs, the target systems are listed beginning with a one 8-bit processor system and continuing toward a sixty-four 64-bit processor system. Actually, there are four separate graphs per page, each showing homogeneous 8, 16, 32, or 64-bit processors, respectively, and each varying in size from one to sixteen processors on a segment. There are three homogeneous systems pointed to by the numbers on the graphs. This representation eliminates the necessity of showing all heterogeneous combinations of target systems. One processor of size M approximates a system that has one processor of size M, plus a combination of smaller sized processors. Likewise, a seven processor system, all of size M, generally approximates a thirteen processor system of which seven processors are size M and six are a smaller size. The approximations referred to are approximations of emulation efficiency, bus speed, and bus latency times.

In Figures 1.3.5.4.1 through 1.3.5.4.3, the emulation efficiency, bus speed, and bus latency times are shown for target systems in which none of the memory accesses are considered shared. The emulation slowdown, the inverse of efficiency, is constant over the range of target systems for a 8-bit and 16-bit machines. The increase in slowdown is due to the limitation of the bus which occurs with large systems. The points marked one, two, and three on the graph represent the worst case heterogeneous target system for each bit class. Note that the values of these points are not significantly different from the single processor system of that class. This holds true on all three figures. The important point to remember about the bus speed graphs is that the speed can never exceed the bus rate, but only asymtotically approach it because the model feeds back bus latency and service times into the instruction cycle time equations.

Note the following bus speed characteristics:

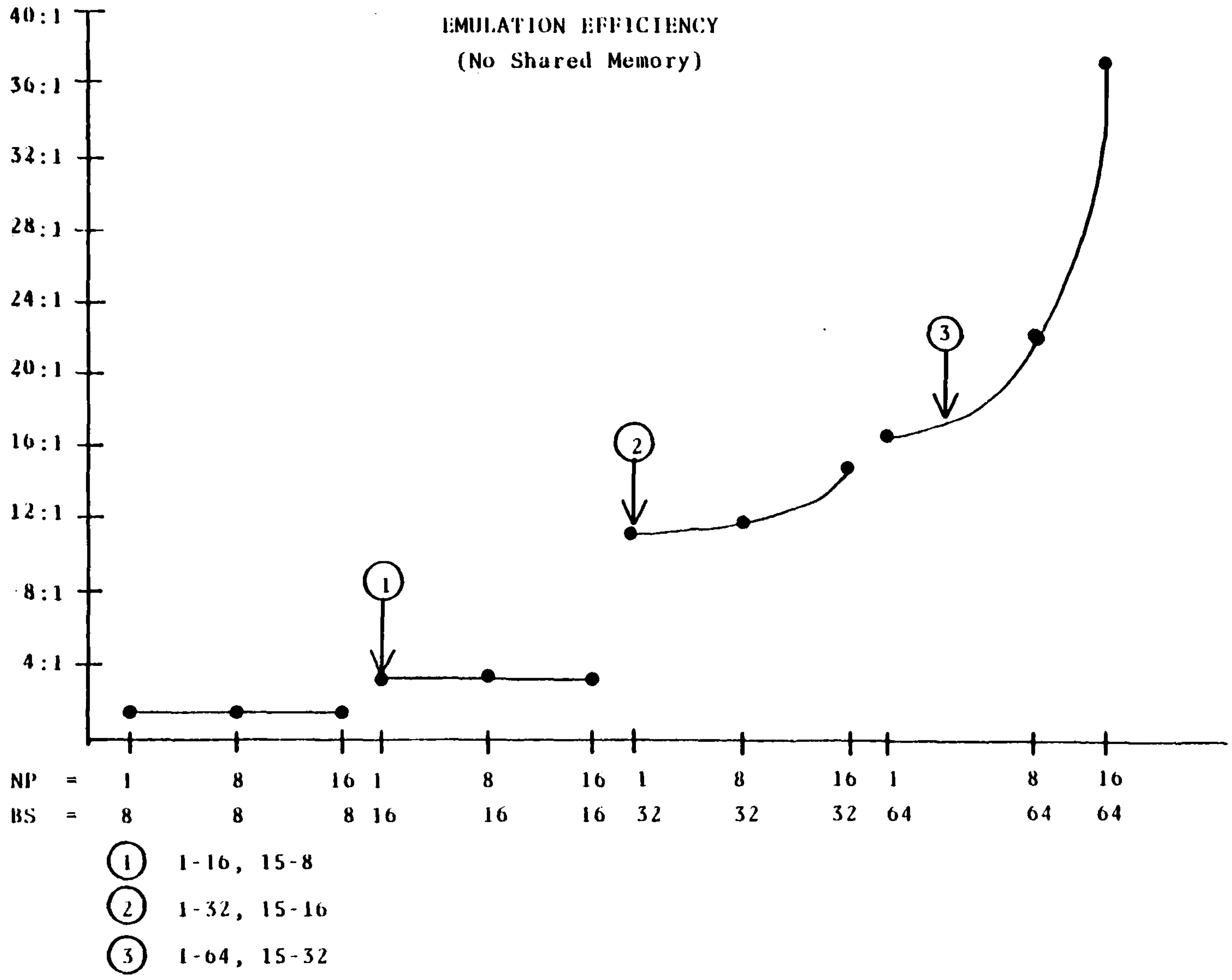
1. Bus speed negliable for 8-bit machine.
2. Bus speed linearly increases for 16-bit machines due to overflow from the local memory.

3. Bus speed quickly approaches the maximum for 32 and 64-bit machines.

Nevertheless, five megahertz is adequate due to the extreme unlikelihood of having a large number of large bit size processors. The rapid increase in bus rate is directly the result of exponentially increasing bus access or latency times.

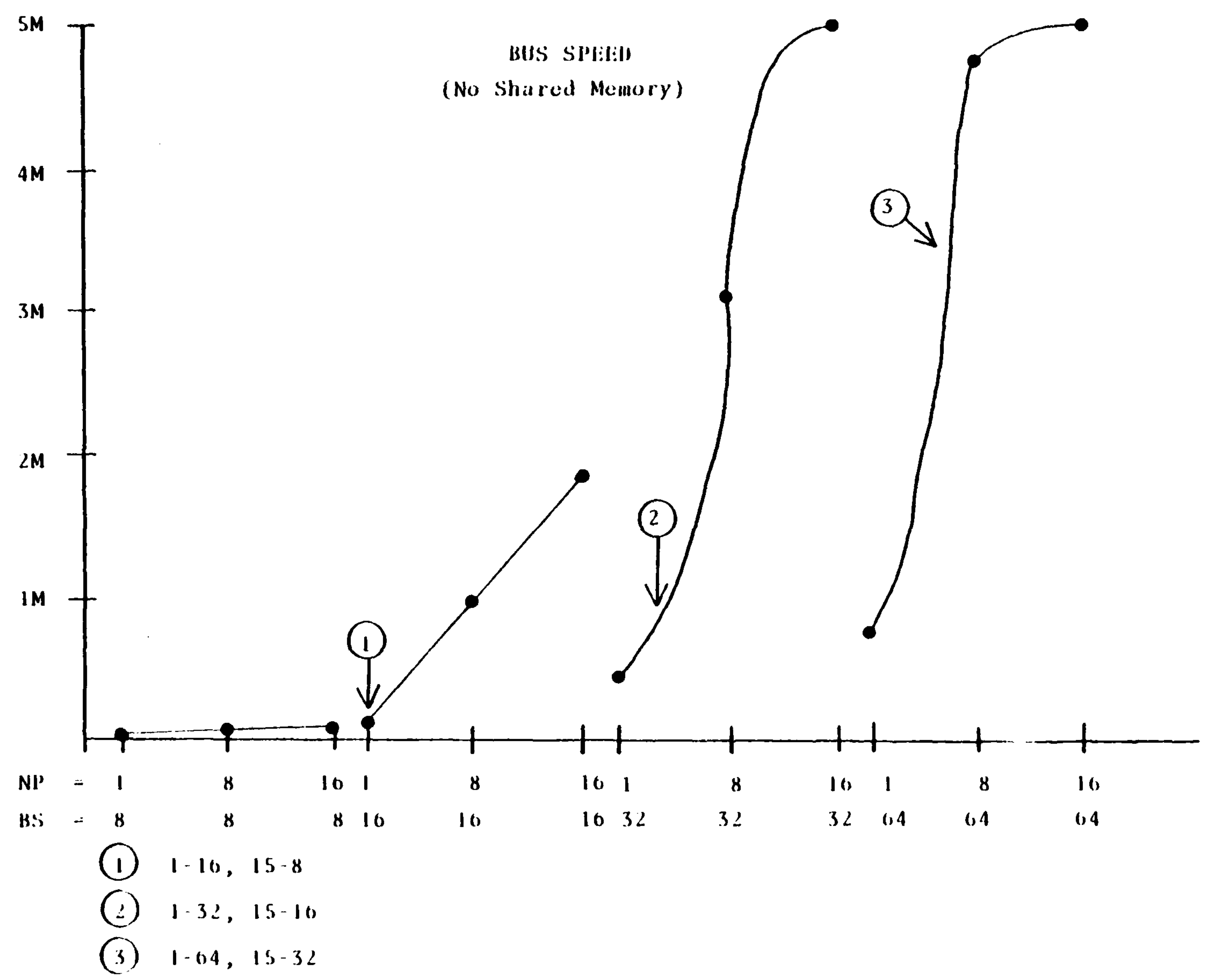
For graphs 1.3.5.4.1 through 1.3.5.4.9, a MMS data bus width of 16 bits was assumed.

Figures 1.3.5.4.4 through 1.3.5.4.6 present shared memory processors with ten percent of their accesses to emulated shared memory. Note that a large number of these smaller processors cause the "slowdown factor" curves to bend upward. This is due to loading of the shared resource controller and its time path considered in the instruction cycle time. The shared resource controller (a microprogrammable processor) was considered to need approximately 15 microcycles to complete one shared request and grant. If each microinstruction takes 200 nanoseconds, then 300K operations can be performed in one second. Note that the shared resource controller's heaviest loading is for 8 and 16 bit machines. The rates for the large machine sizes are well below the maximum. This is due to two major factors. One is that smaller bit size machines generally access instructions themselves in multiple cycles, i.e., op code followed by operand address, whereas large machines generally access the entire instruction at once. Secondly, for each shared access, the machine must complete the request/grant sequency with the shared resource controller. Even though the MMS bus configuration may require multiple cycles for large machines in the emulation run,



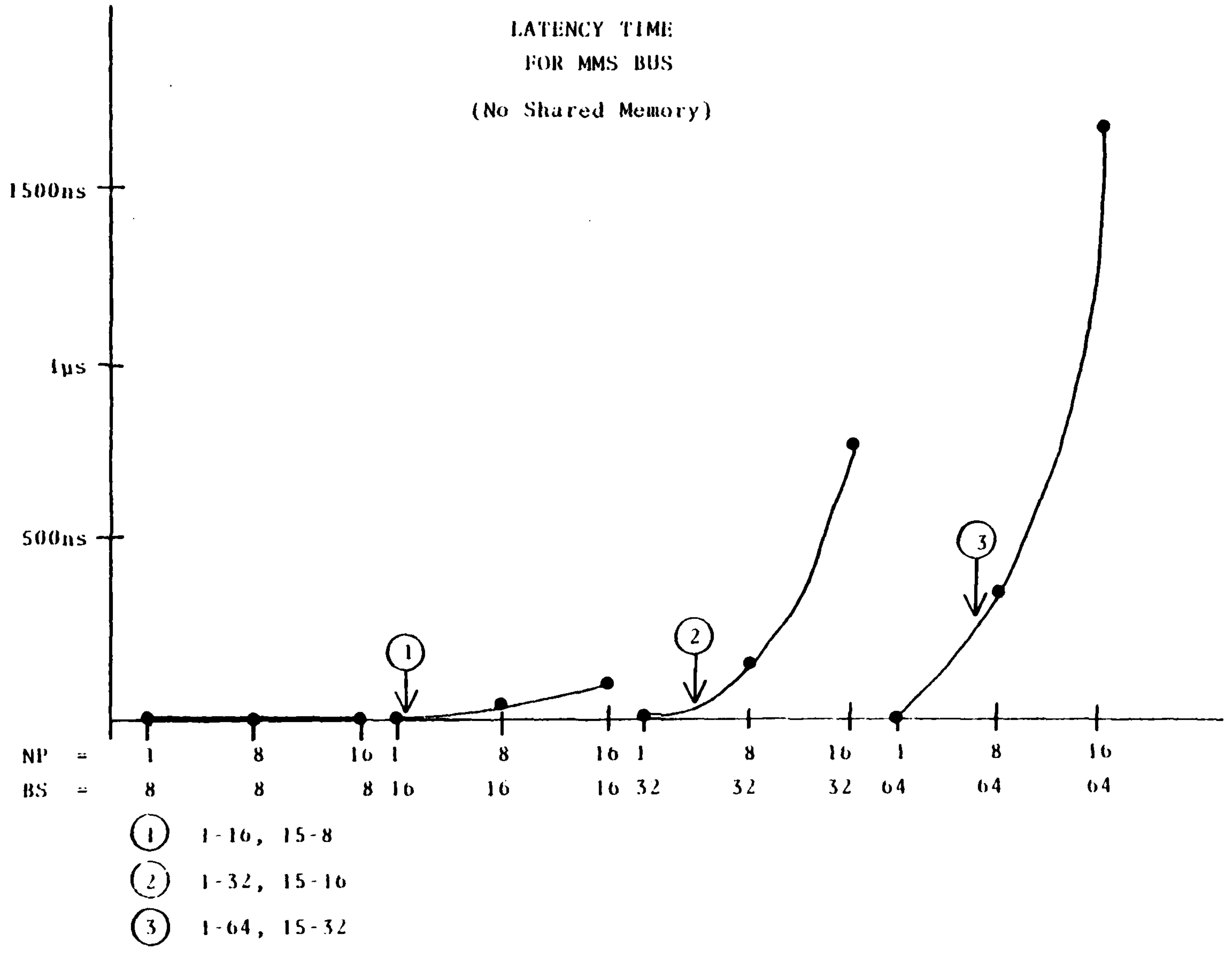
TARGET SYSTEM SEGMENT

Figure 1.3.5.4.1



TARGET SYSTEM SEGMENT

Figure 1.3.5.4.2



TARGET SYSTEM SEGMENT

Figure 1.3.5.4.3

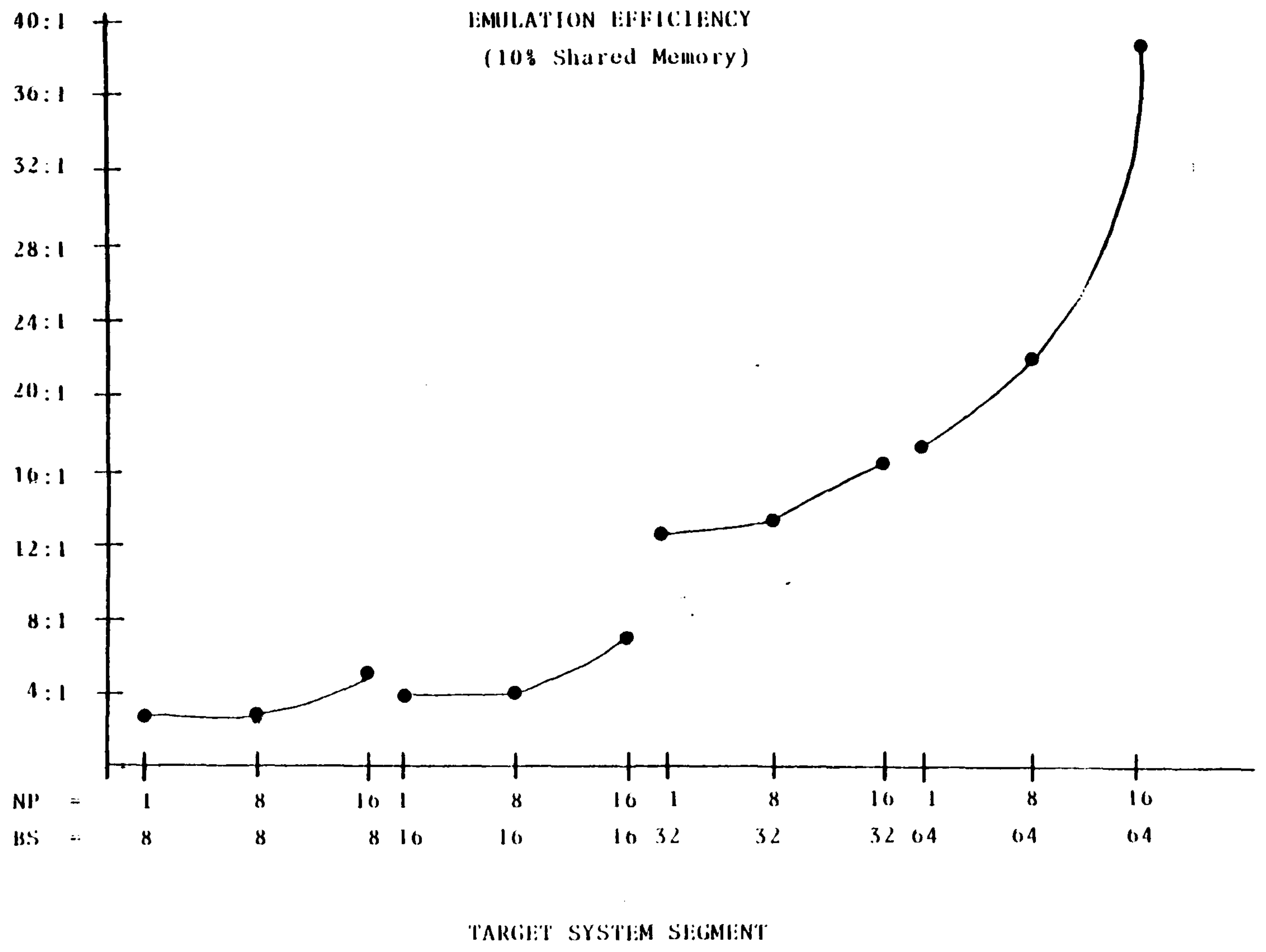
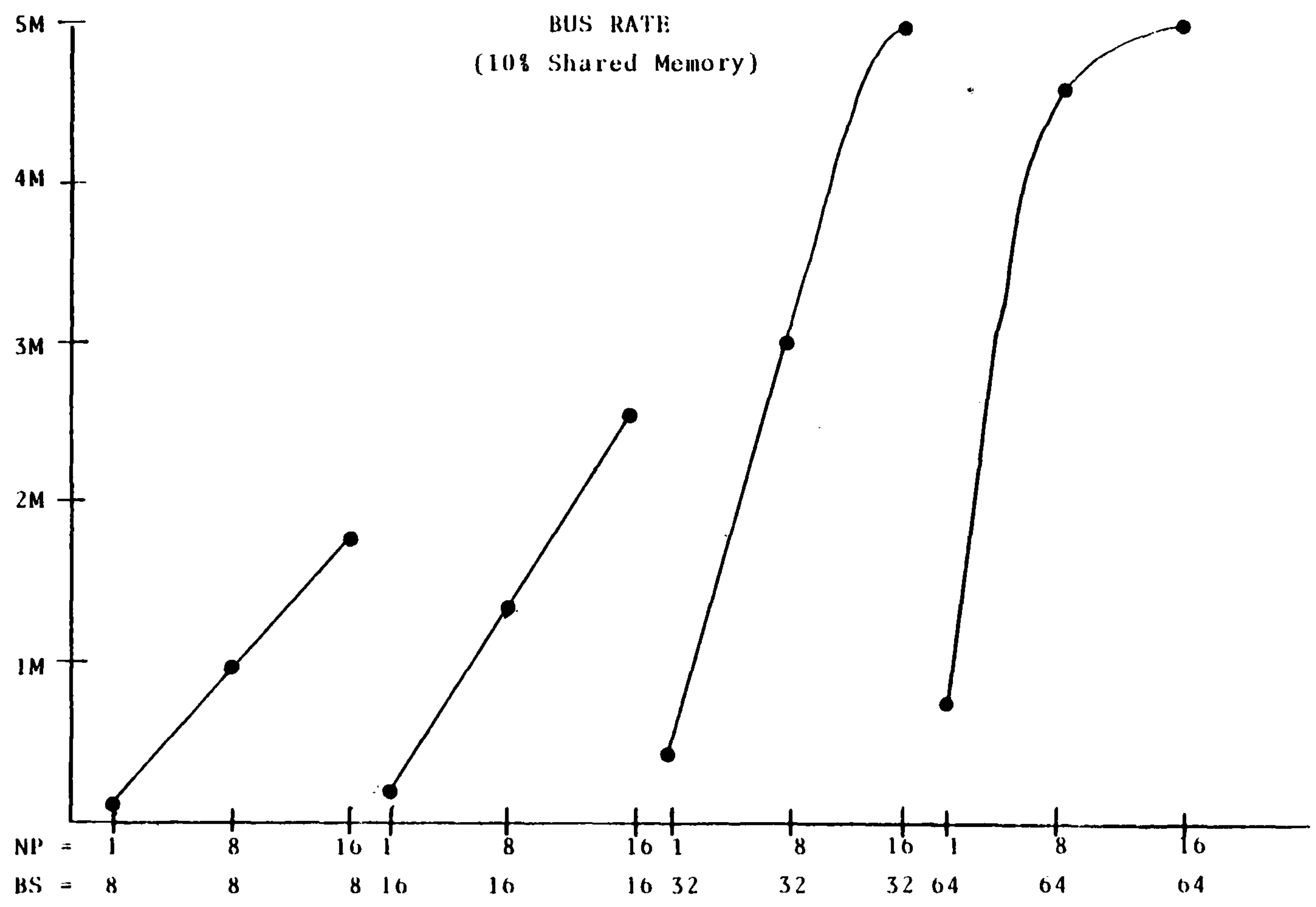


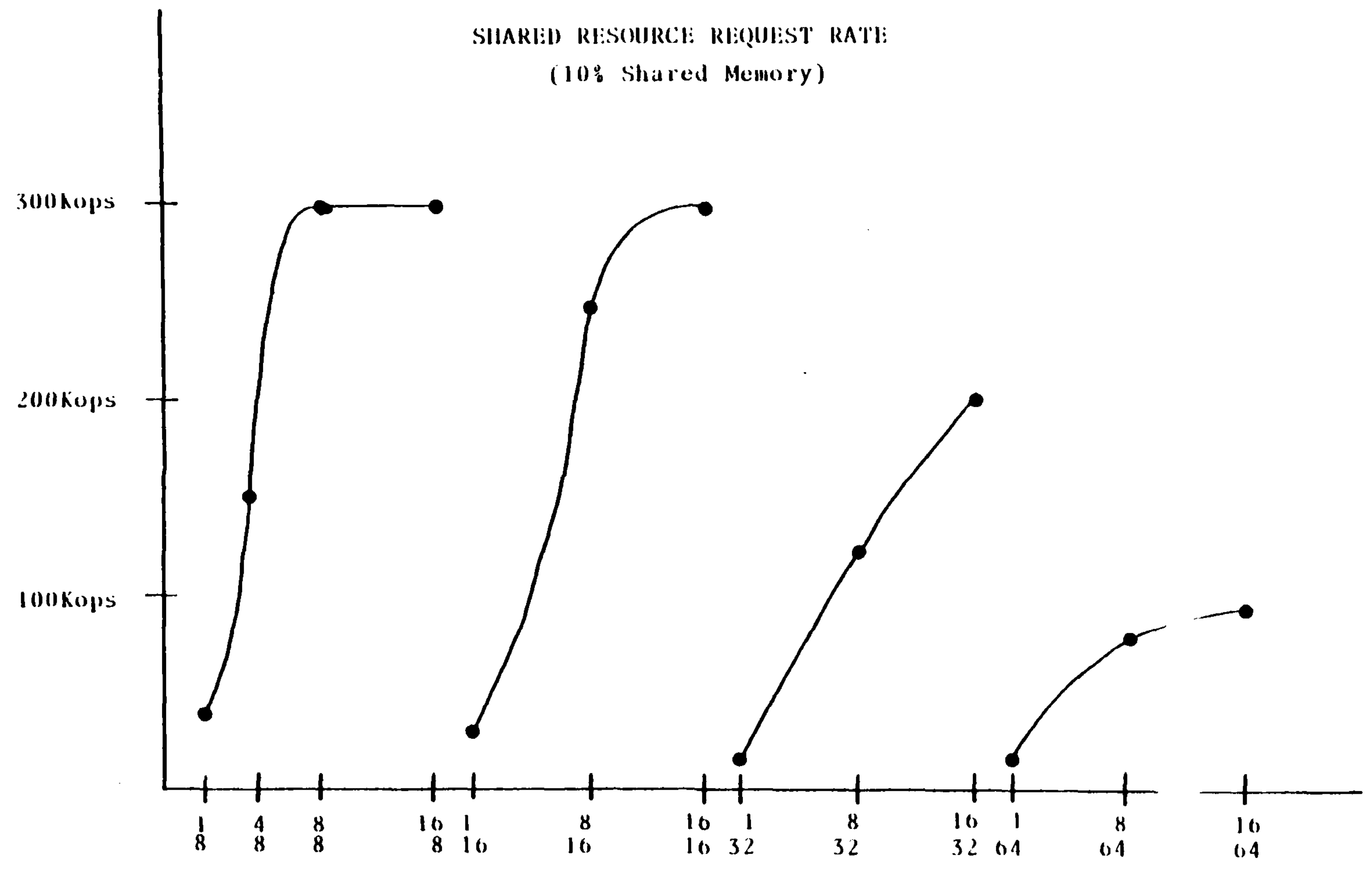
Figure 1.3.5.4.4

used to calculate external action times.



TARGET SYSTEM SEGMENT

F[illegible]ur 1.3.5.4.5



TARGET SYSTEM SEGMENTS

Operation ≡ Service of one shared resource request.

Figure 1.3.5.4.6

only the first of these has a request/grant sequence associated with it. Thus, the larger machine needs to use the bus more extensively. The increase loading of bus results in the bus becoming the system bottleneck. When the bus becomes the bottleneck, the shared resource controller bottleneck doesn't have a chance to effect the system. No analysis has been done with a higher percentage of shared memory in the system. However, it is expected that the emulation would slow down by an order of magnitude or more for an increase in shared memory.

Figure 1.3.5.4.7 shows the sensitivity of the ALU width of the processing element and system data bus width to system performance and chip count in the PE. Since the performance of an 8-bit machine is not greatly affected by these variations, only three graphs were drawn representing 16, 32, and 64 bit machines. The five different MMS configurations plotted on the graphs are 8-bit ALU, 8-bit bus; 8-bit ALU, 16-bit bus; 16-bit ALU, 16-bit bus; 16-bit ALU, 32-bit bus; and 32-bit ALU, 32-bit bus. The PE chip count is arrived at by dividing the PE into three distinct parts: the bus interface unit (BIU), the instruction execution unit (IEU), and the local memory (LM). Each is assigned an approximate number of chips. The chip counts and emulation slowdown ratios for each configuration are as follows.

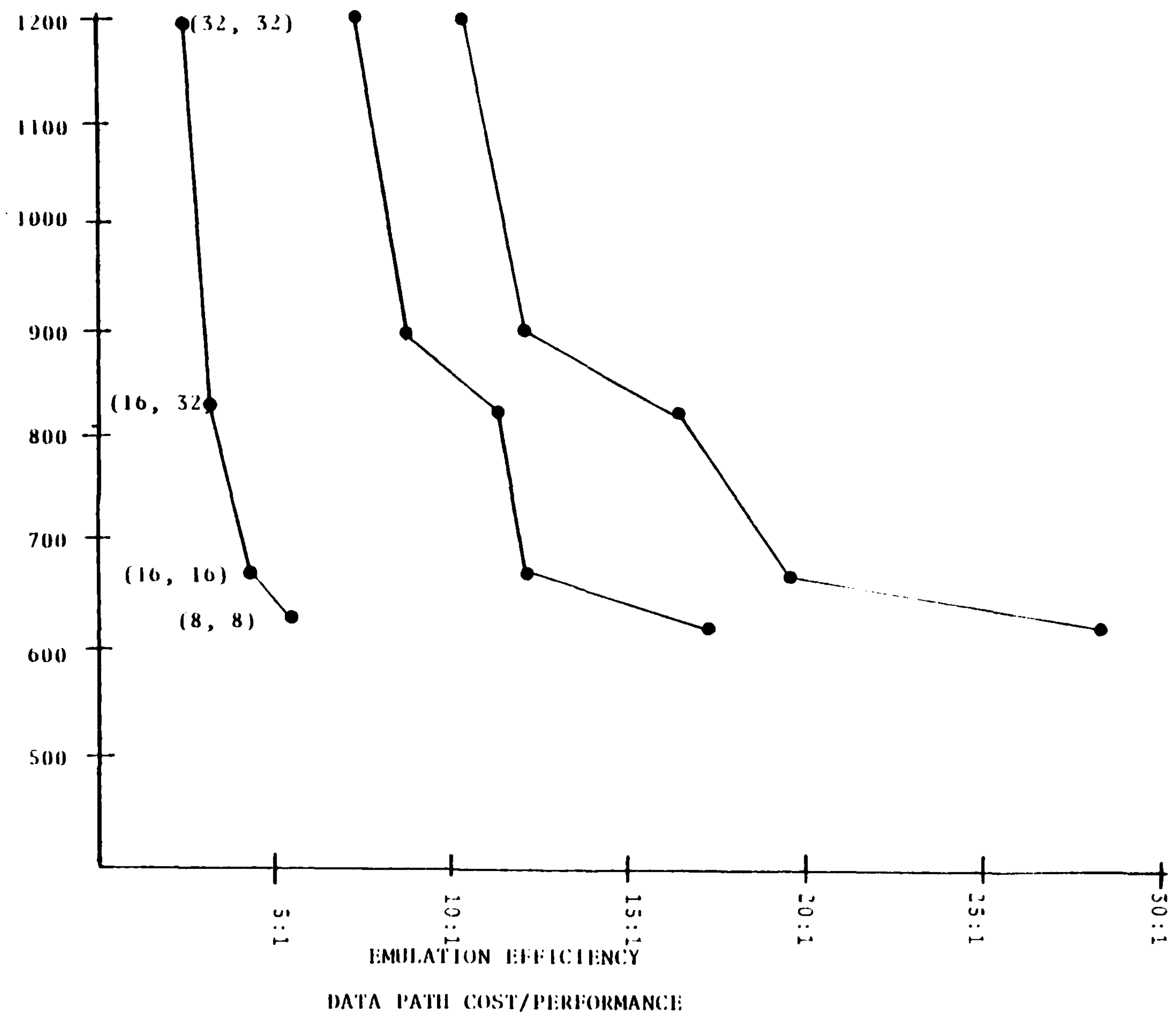


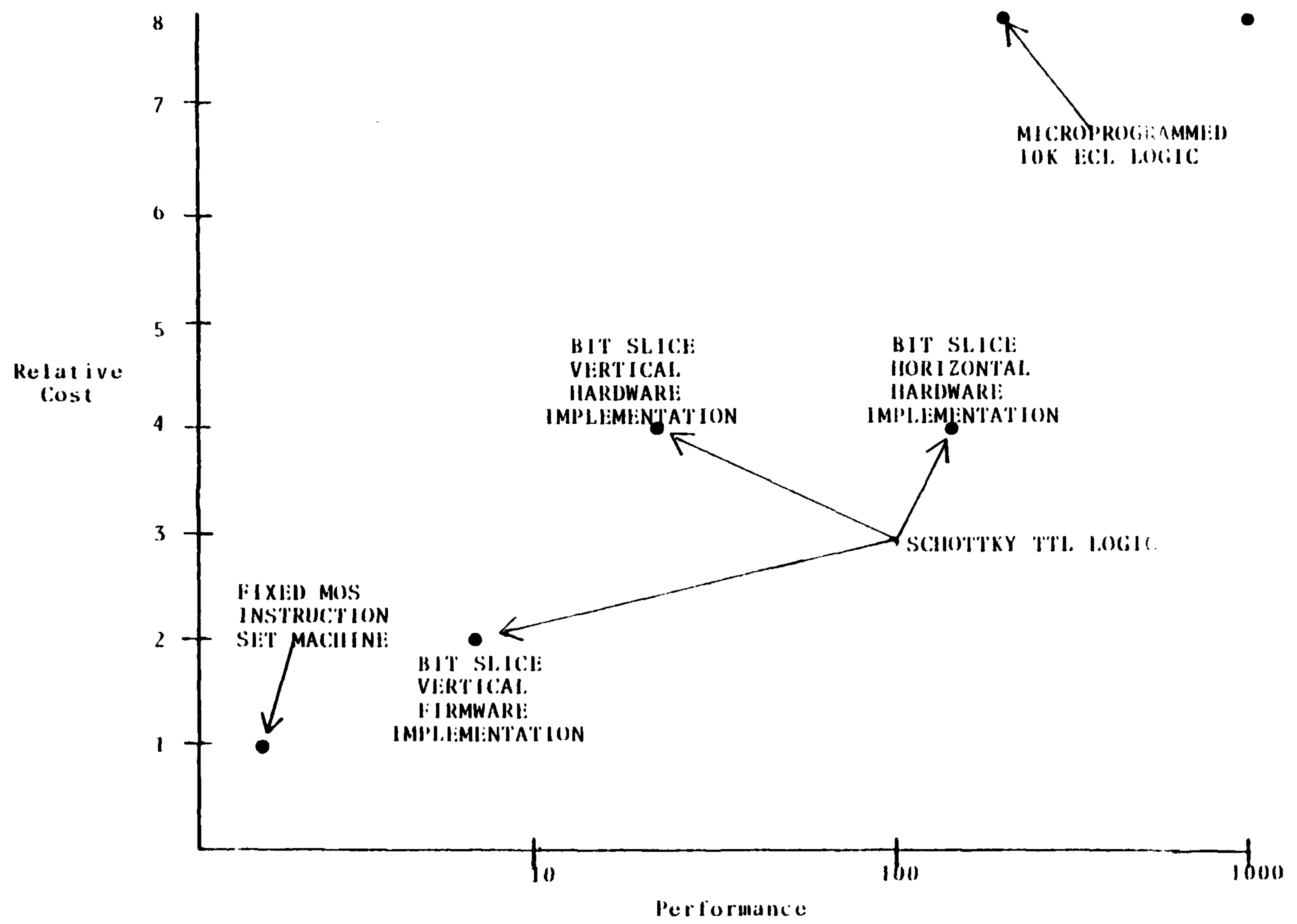
Figure 1.3.5.4.7

MMS CON-FIGURATION		CHIP COUNT				EMULATION SLOWDOWN		
ALUWD	BWD	BIU	IEU	LM	TOTAL	BS=16	BS=32	BS=64
8	8	145	465	20	630	5.25	17.26	28.58
8	16	165	480	25	670	4.22	13.06	19.61
16	16	165	630	25	820	3.61	11.25	16.4
16	32	185	660	50	895	3.41	8.64	11.92
32	32	185	960	50	1195	3.14	7.48	10.46

When choosing the ALU width and bus width, it is desirable to select a point at which the slope of the line is relatively small or at which the slope above the point is relatively steep. The optimal point for the 16-bit machine is at (16, 16), whereas it appears the (8, 16) or (16, 32) point is optimal for the larger bit sizes. It is important to keep in mind here that performance in MMS depends almost entirely on the largest bit size emulated.

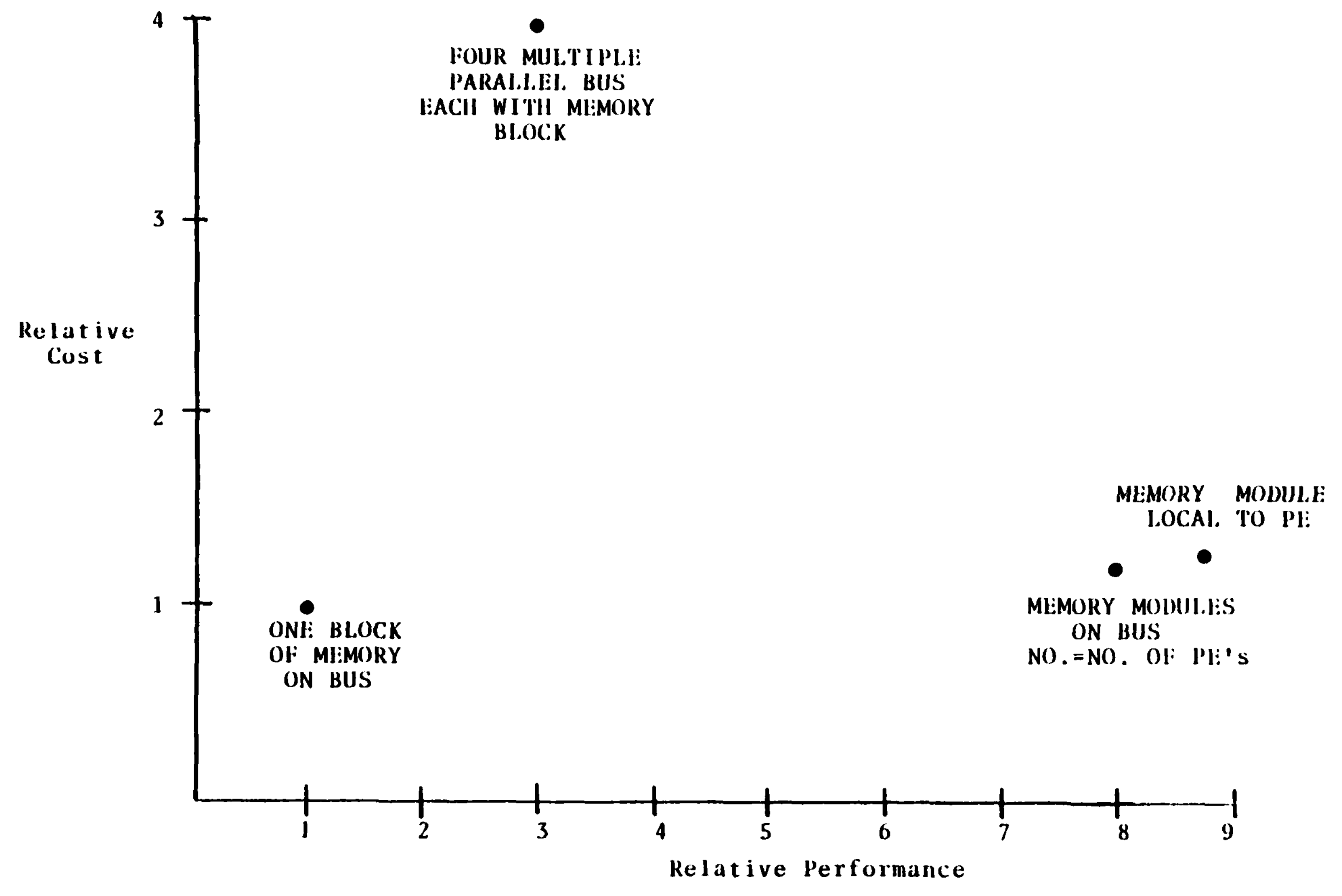
There are other design criteria which may take precedence over this maximal gain/cost criterion. There is certainly a point at which cost may not be exceeded or where a further increase in performance is not worth the cost. However, the (16, 16) or (16, 32) points are presently considered to be in the acceptable range. A more complete analysis is necessary to select the optimum configuration.

Figures 1.3.5.4.8 and 1.3.5.4.9 illustrate the reasons why microprogrammed Schottky TTL logic and a modular local memory organization was chosen for MMS. The cheapest solution is the fixed MOS instruction set machine. These machines are estimated



ARCHITECTURE LOGIC COST/PERFORMANCE

Figure 1.3.5.4.8



MEMORY ORGANIZATION

Figure 1.3.5.4.9

to be twenty times slower than the Schottky TTL bit slice machines. This estimate is based on two factors. First is the lenghtening of the micro cycle time by a factor of about five to ten. Secondly, the loss of parallelism achieved with a wide micro word format decreases performance by a factor of two to three. The 10K ECL solution provides the best performance solution and also the highest cost.

Of the many possible memory configurations, the only real contest was the decision between memory modules local to the processing element or separate on the bus. Since local memories caused no major cost increase, but produced an 8 percent increase in performance and greatly lessened the MMS bus requirements, the local memories were chosen.

Table 1.3.5.4.10 demonstrates what can be accomplished on MMS relative to some well known machines. The major difference in the 8086/Z-8000 types and the PDP-11/45, both sixteen bit processors, is that the PDP-11 is assumed to run at 1 mip while 8086/Z-8000 types run at 0.6 mips in their target system environments. The estimated speed for the 370 was 3 mips.

1.3.5.5 Conclusions

The analysis has shown that a single, wide, parallel bus is sufficient to handle the communication that must exist among processing elements, memory, and other processors in the system when that bus is running at a speed of 5 megatransfers per second and local memories of 128 Kbytes are provided with each PE. The technology chosen for the processors in the PE is Schottky microprogrammed logic. The organization of memory in MMS shall be

INDIVIDUAL EMULATION EFFICIENCIES

OF CURRENTLY POPULAR PROCESSORS

Machine(s)	EE
TMS 9900/LSI-11	1.2:1 (Slowdown)
8080	1.5:1
8086/Z-8000	3.6:1
PDP-11/45	6.5:1
IBM 370	22.5:1

Table 1.3.5.4.10

memory modules which have local access directly from the PE. The cycle times of these modules are independent of the cycle time of the bus.

The I/O processor shall be able to handle interrupts at the rate of one thousand per second with ten thousand per second a desirable goal if any more than one percent of the I/O is environmental in nature or shared between processors. The shared resource controller shall be able to handle requests and grants at the rate of three hundred thousand per second before significantly affecting the performance of sixteen 8-bit or 16-bit processor systems when tne percent of their memory accesses are to shared memory.

The internal paths and ALU width within the instruction execution unit shall be 16 bits wide, while either 16-bits or 32-bits is presently deemed acceptable for paths external to instruction execution unit. Thirty-two bits gives significantly increased performance over 16-bits for relatively little cost, but only when there is at least one 32-bit or 64-bit processor being emulated in the target system. The cost and the emphasis placed on target systems containing only 8 or 16 bit processors govern this decision. A major issue in the costing is the availability of random access memory chips which are 64K bits and are arranged in a 2, 4, or 8 bit wide organization. If a 32-bit organization is chosen, accessed in MMS must be as a 32-bit word as well as a 16-bit or 8-bit element. With the specification of a total of 128K bytes in a memory module, the module must be arranged as 32K X 8 sections to achieve these accesses. If only 64K X 1 chip is available, the upper 32K would not be needed.

The emulation efficiency achieved by MMS, according to the analysis, is excellent in comparison to any simulation executed on a host machine. The factors of slowdown for 8, 16, 32, and 64 bit processors are 1.8, 3.7, 11.5, and 17 to one, respectively. These numbers represent the optimal in design of the emulation by the user and assume that the target system processor is of a reasonably standard and good design.

2.0 THE MMS SYSTEM STRUCTURE

The purpose of this section is to provide a conceptual design of the MMS bussing structure. The topics covered in this section are: the system requirements and the physical operational requirements of the MMS bussing structure, differences in the MMS strawman and the final MMS conceptual design, a functional description of the bussing structure used in the MMS conceptual design, and a functional description of interfaces used by the various devices to interface to the MMS bussing structure.

2.1 MMS as a System

MMS is a system of interactive devices (such as micro-programmable microprocessors, various mini computers, and control elements) used to emulate other multiprocessor systems. The large number of devices and diversity of devices needed to perform this emulation function requires the physical MMS structure to be highly versatile. This highly versatile structure is needed to facilitate the interconnecting of all of the MMS devices and handling of the various types of information transferred between the different devices.

The communication link(s) between the devices in MMS must be capable of interconnecting the MMS control processor and control elements to a minimum of 64 PE's. The following functions require the MMS communications link(s) to be capable of handling multiple information formats: down-loading and up-loading micro-code, collection of PMS data, time alignment of PE's, and inter-processor communications.

In order to make the MMS a useful and cost effective tool, the physical MMS structure needs to have a flexible and modular characteristic. Flexibility allows the most effective MMS configuration to be constructed for each customer. Modularity allows an MMS structure to be expanded when the need for greater processing capabilities occurs.

2.2 Differences in the MMS Strawman and the Present Conceptual Design

The original MMS Strawman shown in Figure 2.1 handles the problems of multiple information formats by using a network of dedicated busses to handle the various types of communications which occur in MMS. The present conceptual design shown in Figure 2.2 replaces the dedicated busses of the strawman which a single unified bussing structure.

The IOC shown in each PE in the Strawman is no longer present in the PE's in the present conceptual design. The function of the IOC is now performed in firmware. This firmware is a microcoded PE executive, PEXE, which is resident in each PE.

The capability to emulate single instruction multiple data (SIMD) and multiple instruction single data (MISD) architectures has been added to the present conceptual design. SIMD and MISD communications are handled by the broadcast controllers shown in Figure 2.2.

2.3 The MMS Bus - A Unified Approach

The MMS analysis revealed that a single high speed parallel bus can be used to handle all of the MMS bus traffic. This unified bussing structure must be designed to operate in an

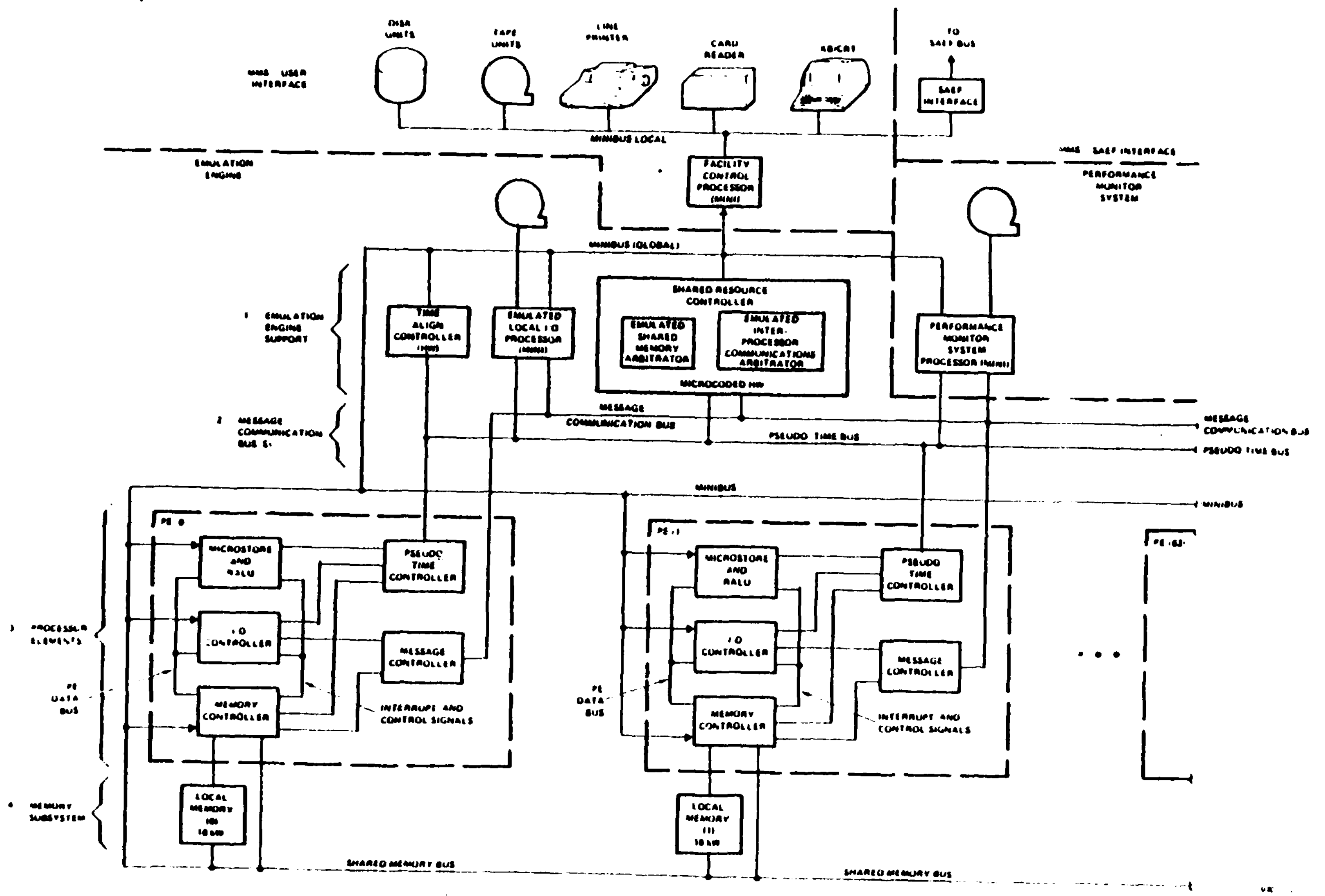


Figure 2-1 MMS Conceptual Hardware Strawman

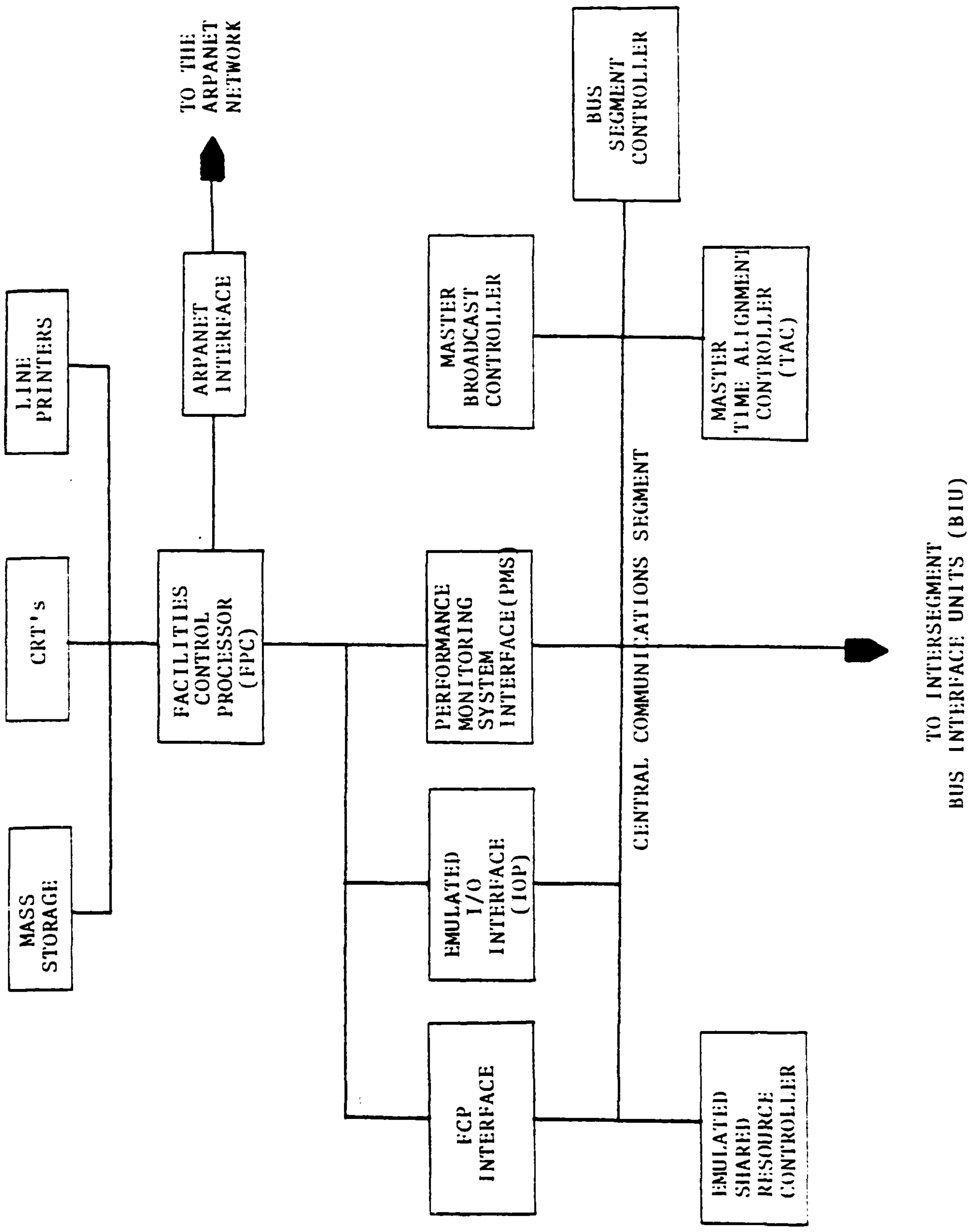

TO THE ARPANET NETWORK
BUS SEGMENT CONTROLLER
LINE PRINTERS
ARPANET INTERFACE
MASTER BROADCAST CONTROLLER
MASTER TIME ALIGNMENT CONTROLLER (TAC)
CRT's
FACILITIES CONTROL PROCESSOR (FPC)
PERFORMANCE MONITORING SYSTEM INTERFACE(PMS)
CENTRAL COMMUNICATIONS SEGMENT
TO INTERSEGMENT BUS INTERFACE UNITS (BIU)
MASS STORAGE
EMULATED I/O INTERFACE (IOP)
FCP INTERFACE
EMULATED SHARED RESOURCE CONTROLLER

FIGURE 2.2.1 THE CENTRAL COMMUNICATIONS SEGMENT

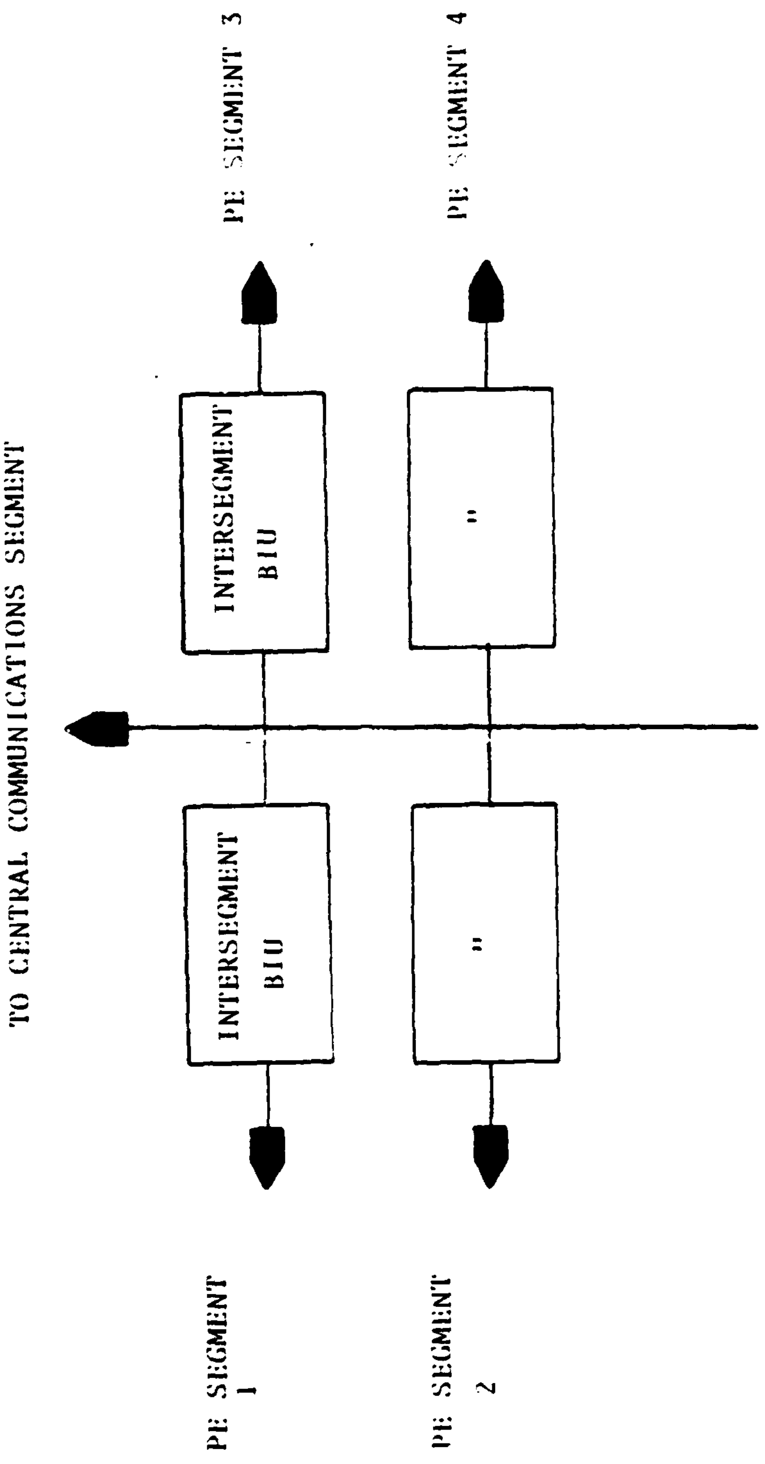


FIGURE 2.2.2 INTERCONNECTION OF SEGMENTS

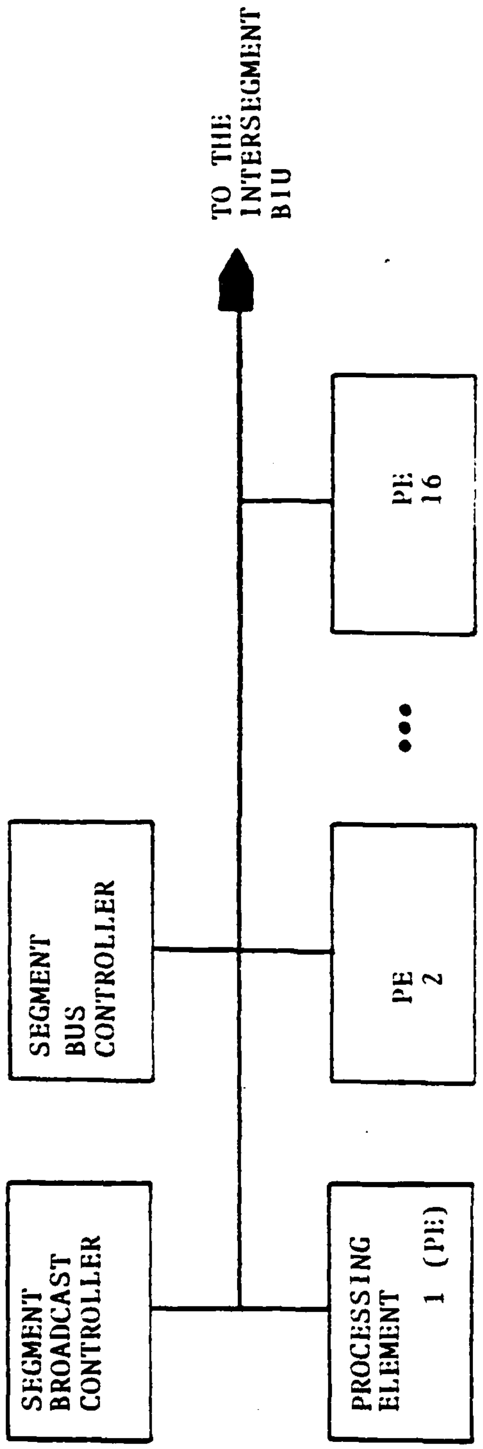
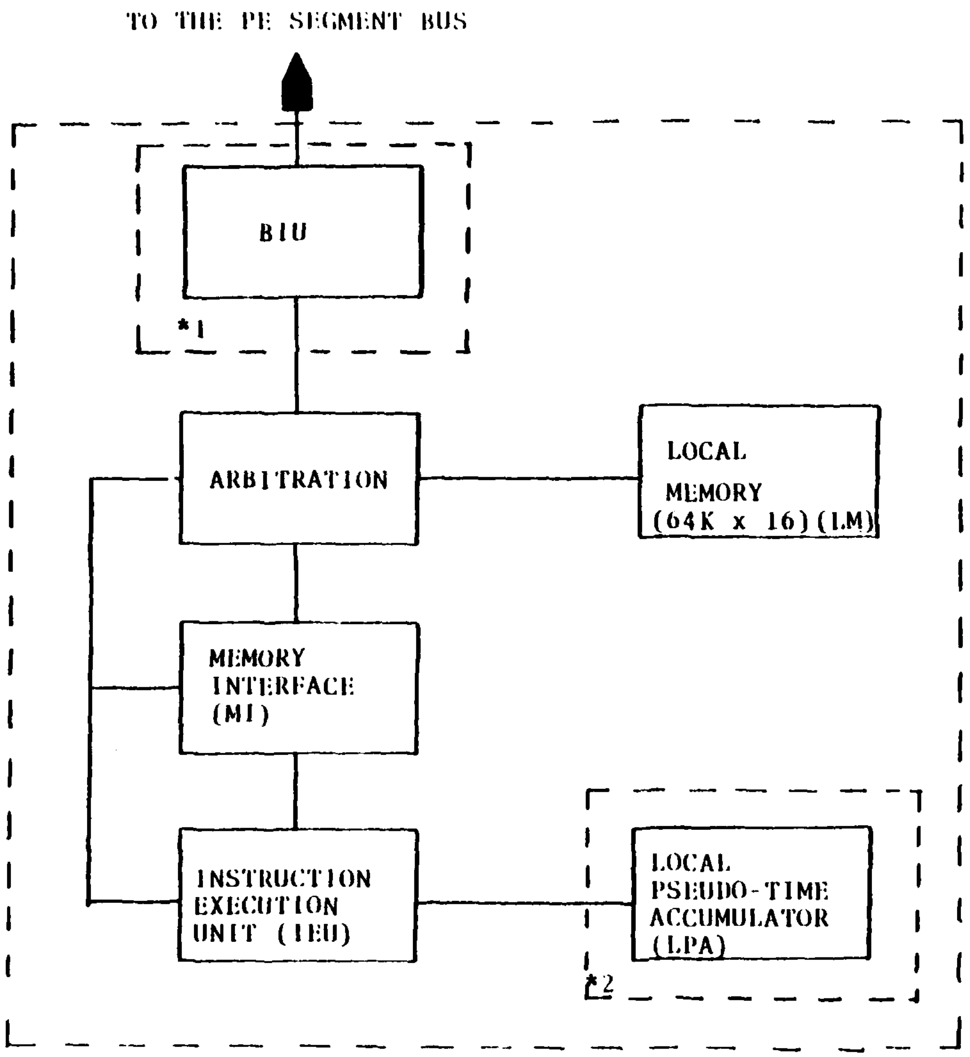


FIGURE 2.2.3 THE PROCESSING ELEMENT SEGMENT


TO THE PE SEGMENT BUS
BIU
*1
ARBITRATION
LOCAL MEMORY (64K x 16) (LM)
MEMORY INTERFACE (MI)
INSTRUCTION EXECUTION UNIT (IEU)
LOCAL PSEUDO-TIME ACCUMULATOR (LPA)
*2

*1 NOTE: The BIU is part of the Communications Bus Subsystem. Every device and interface requires a BIU.

*2 NOTE: The LPA is part of the Time Alignment Controller Subsystem.

FIGURE 2.2.4 THE PROCESSING ELEMENT

environment where the bus is heavily loaded with information transfers and at the same time must contend with interprocessor interferences. Also, it must handle contentions for: memory, I/O resources, and shared resources. To facilitate operating in such an environment, the MMS bussing structure must be an effective processor-to-processor, processor-to-control function, and processor-to-memory interconnect structure.

The communications protocol of this unified MMS bussing structure must effectively utilize bus cycle time in order to handle information transfers at a rate of 1 transfer every 200ns. Bus cycle time includes the time required for sending and receiving devices to transfer information between themselves along with the appropriate transfer acknowledges.

2.4 A Description of SBS Protocol

The high speed parallel bussing structure used in the conceptual design of MMS is called SBS. SBS, Synchronous Bussing Structure, achieves the 200ns transfer rate required for MMS by using the following network protocol:

1) All transfers over SBS are register to register transfers.
2) Bus cycle time and device cycle time are maintained as independent events. For example, when a device sends a read request to a memory, the read cycle is performed "off-line". This allows other devices to use the bus while the read cycle is being performed.
3) Transfer acknowledges and information transfers are

made simultaneous events by deferring the sending of transfer acknowledges between devices to a later bus cycle.

4) If a device receives a "transfer not accepted", the device must request the bus again and resented the information.

2.5 An Architectual Description of the SBS Structure Used in MMS

The unified bussing structure of MMS shown in Figure 2.2 is divided into 5 segments. Segments 1 through 4 each represent the communications link between 16 PE's, a segment broadcast controller (BC), and a segment bus controller/intersegment bus interface unit. Any PE can communicate with any PE on a different segment via the intersegment bus interface units and the central communications segment. It is also used for communications between PMS, FCP, SRC, IOP, TAC, and the master BC, and any processing or control elements on segments 1 through 4.

The actual transfer of information between the processing and control elements via SBS is performed by using 80 parallel lines. This parallel synchronous bussing structure utilizes approximately 31 control lines and 49 information lines per segment. The actual number of lines is dependent on the number of segments and the number of devices interfaced to each segment.

2.6 A Functional Description of the MMS Bussing Structure

The basic unit of the MMS structure is the SBS segment. Each bus segment houses a group of processing and control elements which can function as an independent unit. Communications between

MMS processing elements on the 5 bus segments is performed by using the various bus interfaces and bus control elements.

2.6.1 Interfacing to an SBS Segment

Each processing or control element of MMS is interfaced to a bus segment via a bus interface unit, BIU. The BIU is basically a bank of sending and receiving registers. When receiving information, the BIU must decide if the transfer can or cannot be accepted and send the appropriate "accepted" or "not accepted" response back to the sender. When sending information, the BIU must repeatedly request and access the bus until the transfer is accepted. If the transfer is neither accepted or not accepted, an error condition can be detected by the sending BIU.

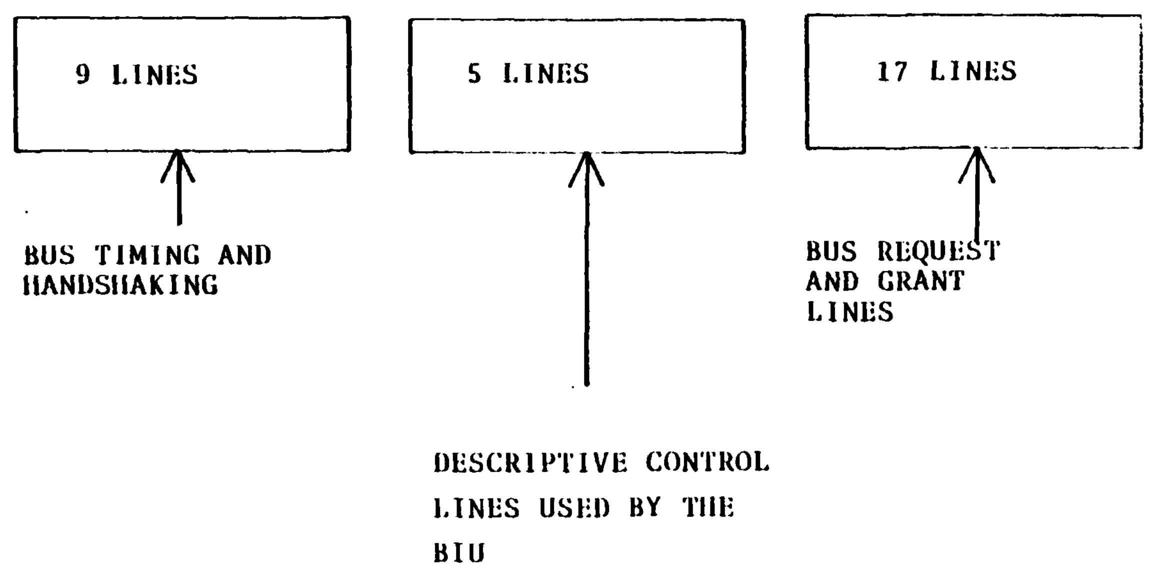
2.6.2 Communicating Over an SBS Segment

A functional grouping of the 31 control lines used on each SBS segment is given in Figure 2.3. The 9 timing and handshake control lines are used for the following functions:

- Valid data on the bus
- Start of a bus cycle
- Transfer accepted
- Transfer not accepted
- Time alignment
- Broadcast communications
- Resetting the bus and the BIU's

The 5 descriptive control lines are used by the BIU's to describe a transfer in the following manner:

- Memory read or write
- Memory hold
- Memory release


9 LINES
5 LINES
17 LINES
BUS TIMING AND HANDSHAKING
DESCRIPTIVE CONTROL LINES USED BY THE BIU
BUS REQUEST AND GRANT LINES

SBS CONTROL LINES

FIGURE 2.3

- A request or a reply
- A word or byte operation

The remaining 17 control lines are bus request/grant lines. Each BIU on each segment has a distinct request/grant line which is used by the BIU to communicate with its segment bus controller. A detailed listing of these 31 control lines is given in Table 2.1.

A functional grouping of the 49 information lines is given in Figure 2.4. The most significant bit of the information field is to specify the parity of the 49 information lines. The second and third bits of the information field are used to determine if a transfer is an interprocessor communication (IPC), broadcast communication (BCC), a local memory address, or a PE internal memory address. The next 8 bits are used to determine to which segment and which BIU on the specified segment the transfer is directed. The format of the remaining 38 information lines is dictated by the type of transfer being performed. These lines can be formatted in a manner which best facilitates the transfer of PMS data, IPC's, BCC's, and memory requests.

2.6.3 Intersegment Communications

Intersegment communications are accomplished via the 4 segment bus controllers/intersegment BIU's shown in Figure 2.2.2. Each of the intersegment BIU's will accept all transfers on its own segment directed to any other segment. When an intersegment BIU receives a transfer to another segment, the intersegment BIU sends the transfer to the appropriate BIU on segment 5. If the BIU is another intersegment BIU, the transfer will be sent to the

TIMING AND HANDSHAKE LINES (9 LINES)

TIC - Time Increment From Time Alignment Controller
SPT - Stop Master Pseudo-time
BCC1 - Broadcast Not Accepted, Hardware Malfunctions
BCC2 - All PE's Ready to Accept Another Broadcast
RESET - Reset Bus
BUS CYCLE - Start of the Time Interval Allocated for a Bus Transfer
ACC - Transfer Accepted
NACC - Transfer Not Accepted
DATA VALID - Valid Data on Bus

DESCRIPTIVE CONTROL LINES (5 LINES)

R/W - Transfer is a Read or a Write
REQUEST/REPLY - Transfer is a Request or a Reply
MH - Allows a PE to Lockout Other Users from a Memory
MR - Allows a PE to Release a Memory from the Hold State
WORD/BYTE - Transfer is a Word or Byte Operation

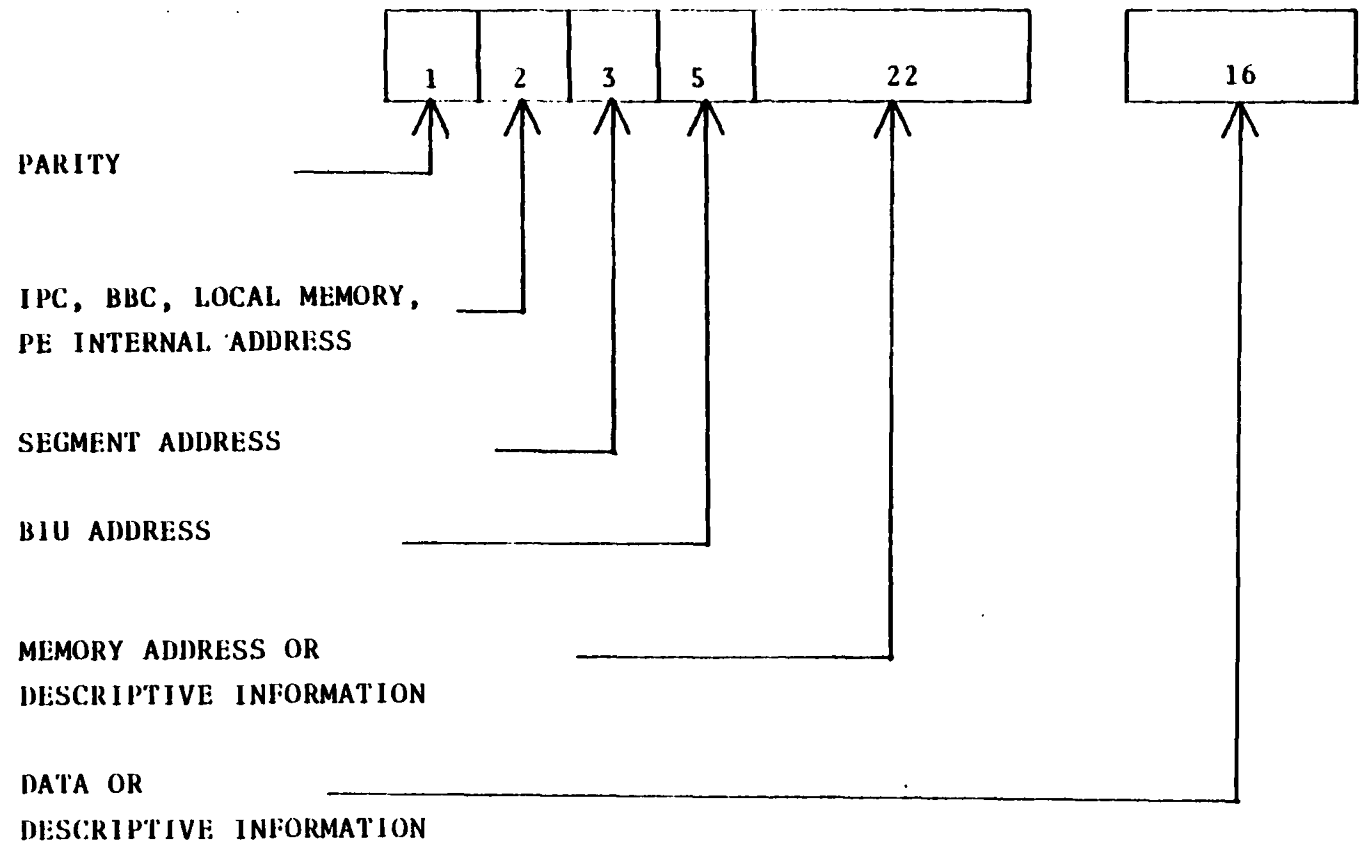
HALF DUPLEXED REQUEST/GRANT LINES (17 LINES)

Each Device on a Segment Bus has its own Request/Grant Line.

INFORMATION LINES (49 LINES)

Carry the Information Transferred Between BIU's.

Table 2.1



SBS INFORMATION LINES

FIGURE 2.4

appropriate BIU on the segment addressed by the original sender.

All intersegment communications are performed in this manner. All handshaking required is done on a transfer for transfer basis between the two interactive devices. No direct transfer acknowledgement is sent between the original sender and final receiver.

2.6.4 Broadcast Communications

Broadcast communications, BCC, are controlled on each SBS segment by a segment broadcast controller. A device wanting to send a BCC must first send the BCC to the appropriate broadcast controller. The broadcast controller will in turn send the BCC to the appropriate devices simultaneously. Broadcast communications are only sent when all devices associated with the sending broadcast controller are ready. For this reason BCC are never present.

Broadcast communications can be performed in any one of the following modes:

Mode 1: Any PE can broadcast to all of the PE's collectively on the same segment via the segment broadcast controller.

Mode 2: Any PE can broadcast to all of the PE's collectively on any one other segment via the broadcast controller of the receiving segment.

Mode 3: Any PE can broadcast to all of the PE's collectively on all of the segments via the master broadcast controller. The master broadcast controller broadcasts to the entire MMS system via the segment broadcast controller.

2.7 Operational and Physical Requirements of the MMS Bussing Structure

The present MMS conceptual design is based on the SBS bussing structure. SBS provides a flexible interconnect structure capable of meeting MMS system requirements and constraints. Also associated with these MMS system requirements are several physical and operational requirements which must be met by the MMS bussing structure. These requirements are as follows:

1) Memory Address Space - MMS will require a memory address space on the order of 256 megabytes (28 address lines) for emulation of very large multi-processor networks.
2) Transfer Characteristic - Most MMS transfers will be single word or byte memory transfers or IPC's.
3) Aggregate Transfer Rate - The MMS analysis revealed the aggregate transfer rate for MMS will be 5 mega-transfers/second
   a. For a wide parallel bus - 50 bits/transfer
      Bit rate = 250 megabits/second
      Clock rate = 5 MHz
   b. For a high speed serial bus - (50 bits + overhead)/transfer
      Bit rate > 250 megabits/second
      Clock rate > 250 MHz
4) Physical Bus Length - The MMS bus will be long enough to interconnect the 64 PE's and also short

enough to allow for a transfer every 200ns.

5) Logic Family - To be compatible with the PE's and interfaces in MMS, the bussing structure will be TTL compatible.

6) Expandability - the MMS structure will have capability of interconnecting more than 64 PE's.

2.8 How SBS Meets the MMS Physical and Operational Requirements

When specifying a memory address, the 49 SBS information lines can be used to address over 1000 megawords of memory. These 49 information lines provide MMS with built-in expansion capability. SBS allows MMS to have as many as 8 bus segments with 32 devices on each segment. This allows MMS to utilize a maximum of 217* PE's.

The SBS protocol is word and byte oriented. SBS provides for only one transfer for every bus request.

Each SBS segment is designed to operate at a 5 megatransfer/second rate. With segments 1 through 4 operating in parallel, SBS provides an aggregate transfer rate of 20 megatransfers/second. To achieve this 5 megatransfer/second rate, each SBS segment is limited to 20 feet, and is designed to interface with the Schottky TTL logic family.

2.9 Alternate Bus Configurations

Before basing the present MMS conceptual design on SBS,

*217 results from using 7 PE segments with 31 PE's per segment + 1 broadcast controller per segment. The 8th segment is for intersegment communications. It is important to note that using these maximum values can reduce emulation efficiency due to changes in physical bus length and bus transfer rate.

serial busses, unibusses, TDM busses, and circulating busses were considered.

Single serial busses are simply too slow to achieve the 250 megabits/second transfer rate of MMS. The performance of a serial bus can be improved by using multiple serial busses. However, the improvement is overshadowed by the cost and complexity of interfacing to a multibus architecture. The estimated cost for using Hyperchannel, a multiple serial bus network, in MMS exceeds 1 million dollars.

The possibility of using a Unibus concept is thwarted by the physical length of such a bus. If the 64 PE's could be interconnected with a 100 foot bus, the worst case cable delay would be 200ns. This delay alone would be equivalent to the time required to perform a single MMS transfer.

The relationship between the physical bus length, number of processors to be interconnected, and the 200ns transfer rate are among the primary reasons for using a segmented bus architecture.

Alternatives to the segmented parallel SBS bus are parallel Time Division Multiplexed busses (TDM) and parallel circulation busses. However, these types of bus structures are plagued with the same problems as the multiple serial bus structure. The bus interface units for the TDM and circulating busses are highly complex and expensive. These bussing structures are generally designed for applications where large blocks of data are being transferred between devices.

Also associated with TDM and circulating busses are problems, such as poor access time, difficult loop expansion, poor acknowledge time, and transfers getting trapped on a loop.

2.10 Details of the MMS Interface for SBS

All devices to be used in MMS must be interfaced to SBS via a Bus Interface Unit (BIU). All of the BIU's needed in MMS are similar to the PE BIU. Each BIU contains a send and receive function which operates independently of each other. The control function in the BIU is required to perform the following functions:

1) Decode the BIU address lines and type of transfer.
2) Count bus cycles to know when to send or receive transfer acknowledges.
3) Generate transfer acknowledges and the data valid signal.
4) Detect when no acknowledge is received.
5) Send bus reuqest and receive bus grants.
6) Generate a parity bit before sending and check parity when receiving.
7) Send error messages to the IEU or FCP.

2.10.1 The Segment Broadcast Controller

The segment BC controls broadcast communications with two "wired or" lines which are connected to each device on the segment. Line BCC1 indicates to the broadcast controller that all devices are ready to receive a broadcast communication. If a device fails to accept a broadcast when BCC1 indicates all devices are ready, BCC2 indicates to the BC a hardware failure has occured in one of the BIU's.

2.10.2 Segment Controller/Intersegment BIU

The segment controller/intersegment BIU is essentially a PE BIU used to interconnect a segment of PE's with central communication segment 5. This interface also contains the additional hardware to perform bus arbitration and generate the bus control timing.

2.10.3 The Facilities Control Processor Interface

During system initialization and debug, the FCP interface can perform the following special functions:

1) Down load blocks of data - by loading the FCP interface with an MMS starting memory address, a FCP starting memory address and a transfer count, the FCP interface can move the contents of a block of FCP memory to a specified MMS memory block.
2) Up loading blocks of data - by loading the FCP interface with an MMS starting memory address, a FCP starting memory address, and a block count, the FCP interface can move the contents of a specified block of MMS memory to the FCP.
3) Verify block of data - by loading the FCP interface with an MMS starting memory address, a FCP starting memory address, and a block count, the FCP can compare the contents of the two memory blocks.

2.10.4 The Performance Monitoring System Interface

The functions of PMS have been divided into hardware functions and software functions. A detailed discussion of these functions is found in Section 6.

The PMS functions performed in hardware are implemented in the PMS interface, PMSI. This interface requires registers, counters, comparators, high speed memory, and the required contron circuitry to perform the following PMS functions:

1) DMA PMS data to the FCP processor's memory - PMSI utilizes several registers and counters to DMA all PMS data to two buffer areas in the FCP processor's memory. When one buffer is full, PMSI interrupts the FCP processor to move the full buffer to off line storage. At the same time the PMSI can begin filling the second buffer with data.

2) Detect real time events - high speed memory is used to store a set of events the user desires to monitor during run time. All incoming PMS data is compared against these real time events. When a real time event is detected, PMSI interrupts the FCP processor to read the data from the interface.

3) Stop master pseudo-time - when PMS data is being sent faster than the PMSI can service the bus, master pseudo time is halted until all the pending PMS events can be serviced.

4) Detect the start of an event or a periodic event - PMSI uses several registers, counters, and comparators to determine when an event should occur with respect to master pseudo-time. When the time for the start of an event is detected an interrupt is sent to PMS Processor. The PMS

Processor can then perform or service the event.

5) Insertion of time tag to PMS data - PMSI uses several registers, counters and comparators to determine when to insert a PMS time tag.

3.0 CONCEPTUAL DESIGN OF THE PROCESSING ELEMENTS

The conceptual design for the main computing element of the emulation engine, the processing element (PE), is described in this section. The PE design is the result of multiple system tradeoffs conducted during the study relative to the implementation of processor emulation, I/O emulation, memory emulation, execution time emulation and performance monitoring. These tradeoff evaluations have been made with respect to the major design features of MMS: cost, ease of use, flexibility, and performance.

The interim functional performance specification listed four major functional components within the PE:

- Instruction Execution Unit (IEU)
- Memory Interface Unit
- I/O Controller
- Local pseudo time accumulator

Performance goals dictated, almost from inception of these functional components, that the instruction unit, memory interface unit and local pseudo time accumulator be implemented in hardware.

The main question to be resolved is the implementation of the I/O controller (IOC). Alternative considered for implementation of the IOC are:

1. A separate bipolar processor that is microcoded like the instruction execution unit (IEU) of APE.
2. A separate MOS processor of the Intel 8086 category.
3. A combination of the IOC and IEU hardware with appropriate shared software execution.

The third alternative, a combined IOC/IEU is considered the best choice for implementation in MMS. The rationale for this decision is as follows:

- Ease of Use - A software interface is provided between the IEU and IOC, which is vendor supplied rather than user supplied. This software interface can be made more user oriented and easier to manipulate than a hardware interface.
- Cost - Lowest of three alternatives.
- Performance - The combined IOC/IEU is performance ranked slightly below the separate bipolar IOC processor and well above the separate MOS processor alternative. Any performance impact is expected to be small since I/O emulation code executes at a low rate (one I/O access per 100 instructions on the average), and contention for the processor between I/O emulation tasks and instruction emulation tasks is expected to be minimal.
- Flexibility - The software IOC/IEU interface is more powerful than a hardware interface and thus more flexible.

3.1 PE Software Conceptual Design

A conceptual design for PE software is included in this section of the report because software is considered as important as PE hardware with respect to the MMS design features of cost, ease of use, flexibility, and performance. It is intended to provide an integrated design that is a "good" hardware/software implementation for performing emulations.

3.1.1 PE Software Design Goals

One of the major design goals of the MMS study relative to PE software has been to obtain modularity and reusability of emulation code. As in any large processing system, structure and modularity of executable code is critically important. A number of programmers are expected to be coding the micro machines within the emulation engine, and the resultant code modules must interact properly among themselves and also with the system code provided in the facility control processor, PMS processor, I/O processor and other programmable resources within MMS.

Reusability of code modules is also considered critically important. Many of the processors and I/O devices that are present in any one target system are also present in other target systems of interest. It can be of great benefit to the users of MMS if software developed for prior target system emulations can be captured and reused if the same target processors and/or I/O are reused.

In order to derive maximum benefit from reusable emulation code, it is also necessary that processor emulation code and I/O emulation code be separable. This separation is necessary to keep the emulation code from being excessively target system dependent. If indeed this separation can be attained, then user effort can be directed toward I/O emulation which is most likely to be target system dependent, and away from processor emulation which is least likely to be target system dependent.

In a like manner, it is also desirable to attain separation or modularity of I/O emulation code. Since physical I/O devices within a single target processor rarely communicate among themselves, the modularity of I/O emulation code should be straight forward to attain. This necessitates that a standardized communications structure be implemented between I/O emulation code and instruction emulation code.

3.1.2 PE Software Organization

When PE software is organized in a modular fashion, instruction emulation code and I/O emulation code exists as separate modules. The I/O emulation code can be divided into modules according to the emulated I/O device type.

The PE executive (PEXE) provides the linkage mechanism between the various emulation code modules. PEXE is a vendor supplied software module that provides resource allocation, macro interrupt handling, a communications structure, and other additional support functions. PEXE is essentially a small real time executive. It exists here for precisely the same reasons that real time executives exist in any computer system.

The PE software organization presented here meets all the design goals discussed in the previous paragraphs. Its primary advantage over other alternatives considered during the study lies in a large reduction of user effort required to produce emulation code. The ease of use gain that is associated with the vendor supplied executive is sufficient justification for its inclusion in the MMS structure.

3.1.3 PE Executive (PEXE)

The components of PEXE and "intra PE" functions are described in this section. The interaction of the executive with other MMS system software such as the system generation language is not discussed in this section. Rather, the systems aspects of the executive are included in the software conceptual design.

PEXE is a small real time executive, resident within each PE of MMS. Its major purpose is threefold:

- o Coordinate and link instruction emulation code with I/O emulation code.
- o Coordinate and Line I/O emulation code with external processor (physical I/O through the MMS Bus), and
- o Provide common support functions.

The major components or functions that have been identified as being within the domain of PEXE are listed below. Each is discussed in detail in the following paragraphs.

- o Micro Interrupt Routing
- o Software Interval Timing
- o I/O Instruction Routing
- o Macro Interrupt Scheduling
- o DMA Scheduling
- o Macro Code Debug Support
- o PMS Support
- o Error Handling

3.1.3.1 Micro Interrupt Routing

Micro interrupts are utilized within the PE for scheduling of physical I/O operations in much the same manner as macro interrupts are utilized with typical macro processors. Sources of micro interrupts within the PE can be categorized into the following groups:

- o Interprocessor Communications (IPC)
- o Broadcast IPC
- o Interval Timers (Pseudo time)
- o Error Conditions

- Performance Monitoring (PMS)

The first two groups (IPC) are used for all physical I/O external to PE. This I/O conducted through the MMS bus which provides a communications path to all other resources of the emulation engine. The second two groups represent conditions internal to the PE. A bank of interval timers with master pseudo time as the input reference is used to support I/O emulation. Error conditions that may occur asynchronously with the PE execution, such as a bus time out, are also sensed by micro interrupts. The last group of micro interrupts service the performance monitoring system.

The primary function of PEXE with regard to microinterrupts will be to route control of the processor (IEU) to the various interrupt handling routines. These handlers will exist primarily as I/O emulation code although some interrupt handling, particularly error conditions and PMS, will be performed by the executive itself. The executive will perform microinterrupt routing via table lookup of code entry points. This table will be located in the internal memory of the PE. PMS support will be provided for all of these entry points except error and PMS conditions so that extensive performance monitoring of I/O operations can be achieved.

In addition to microinterrupt routing, the executive will also be responsible for scheduling of the processor itself. This scheduling will be rather simplified since extensive nesting of tasks will not be required.

3.1.3.2 Software Interval Timers

The bank of hardware interval timers within the PE is expanded by a software interval timer routine within the executive. This is intended to provide a soft rather than hard limit on the number of interval timers available for I/O emulation. The software interval timers are particularly applicable for low rate I/O such as terminals.

3.1.3.3 I/O Instruction Routing

Routing of I/O accesses is the primary communications method used in the executive for linking I/O emulation code with instruction emulation code. This linkage is supported by the macro interrupt and DMA scheduling tasks described below. I/O access routing is performed by table lookup through the internal memory of the PE entry points in I/O emulation code. These entry points are supported by PMS for extensive I/O performance monitoring.

3.1.3.4 Macro Interrupt Scheduling

Scheduling of macro interrupts is a common support function provided by the executive. It is implemented by a routine callable from instruction emulation code to remove interrupts from the queue.

3.1.3.5 Macro Code Debug Support

The executive includes a general purpose macro code debug package that controls instruction execution and provides visibility into the emulated macro machine from the PMS processor. The macro debug support is discussed in more detail in the performance monitoring system design.

3.1.3.6 PMS Support

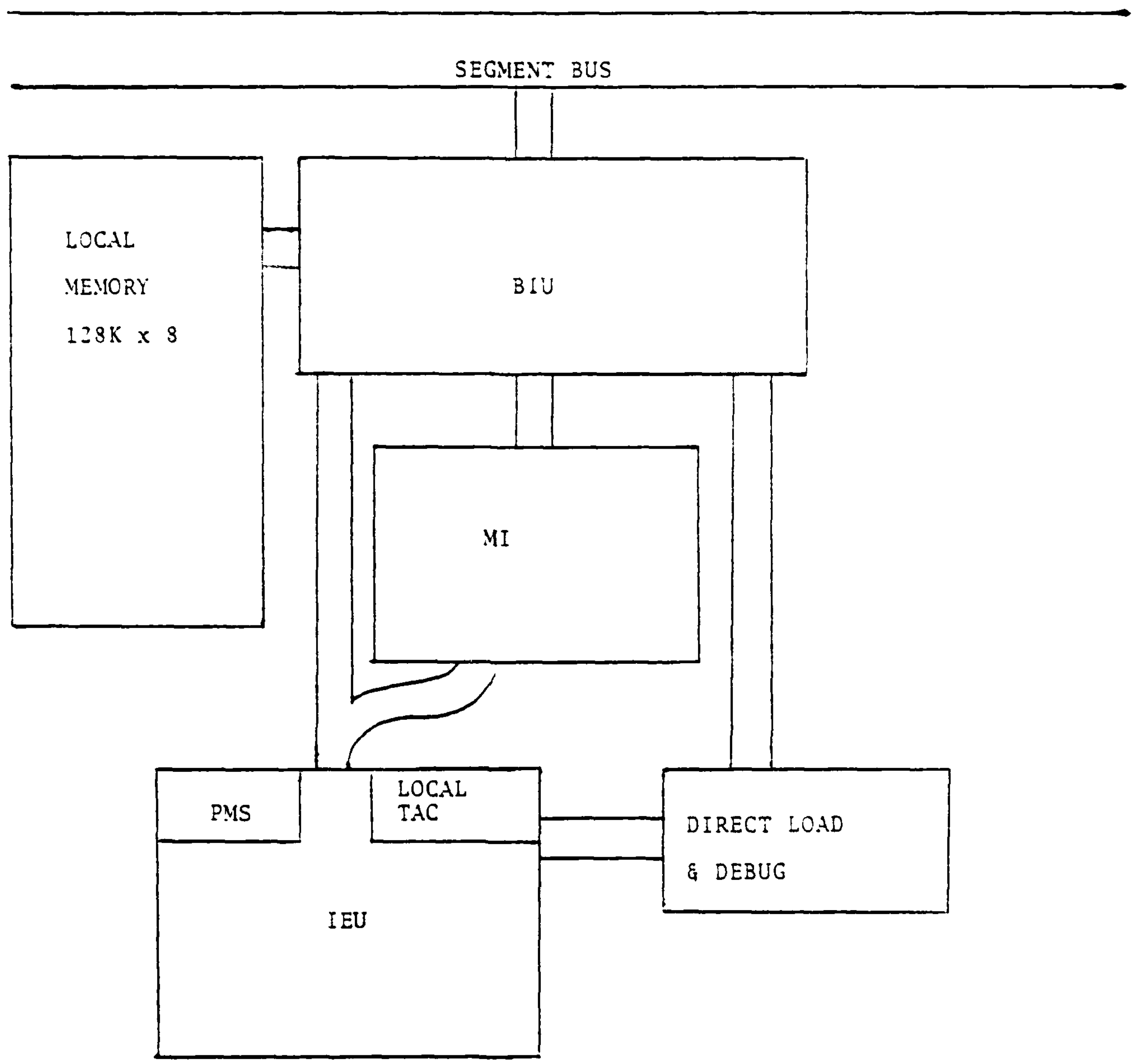
An extensive amount of performance monitoring support is provided by the executive to reduce user burden associated with PMS and thus to increase the ease of use feature of MMS. PMS support is implemented by an interrupt driven routine in PEXE. PMS interrupts are generated by a flag bit contained in each location of internal memory. The PMS support routine dumps performance data to the PMS processor. Further discussion of PMS support is contained in the performance monitor conceptual design.

3.1.3.7 Error Handling

A modest amount of error handling is provided in the executive to notify the facilities control of error conditions which occur within the PE. Error conditions can result from the MMS hardware such as bus time out or parity error, or from the emulated target system, such as access of non-emulated memory.

3.2 Processing Element Hardware

The processing element includes all the logic necessary to emulate a single processor from a target system, enough memory space to handle a reasonably sized program (including many operating systems), and all logic necessary to perform communication with other PE's. In certain cases the PE may be able to handle the complete emulation of the computer (inclusion of I/O, interrupt emulation) while in other cases where separate processors are necessary in the target system to implement a computer, it is probable that two or more PE's may handle the emulation. In Figure 3.2, a block diagram is shown of the processing element. Discussion of the PE hardware is divided and elaborated upon in four parts. The first and largest part is the IEU. Within the discussion of the IEU, PMS, local time alignment control, the direct load, and debug functions are presented because these functions (as well as I/O emulation) are implemented within the hardware of the IEU. The microword format is discussed in the second section, while the third and fourth sections cover the memory interface (MI) and the local memory module (LM) respectively.



APPROXIMATELY 820 CHIPS

PE

FIGURE 3.2

The BIU is discussed in detail in the system bus section.

3.2.1 Instruction Execution Unit

The instruction execution unit is the processor and associated hardware within the PE which handles the emulation of a target system processor. It is within this section of hardware that the functions of instruction fetch, decode, and execution, as well, as I/O emulation support and MMS functions of time alignment, debug, and performance monitoring are performed.

3.2.1.1 Sequencer

The sequencer for microinstructions will be of the 2900 variety. The decision remains as to which of the 2909, 2910, or 2911 chips should be used. The 2910 and 2911 are prohibited from loading their address register while direct inputs are being used and the 2901 is limited with a fixed instruction set. However, the 2909 and 2911 will require additional hardware to supplement its instruction set, and require three chips instead of one to sequence through 4K of microcode. The decision will be based on the needed flexibility to interface with the detailed logic used in instruction execution.

3.2.1.2 Instruction Decode Logic

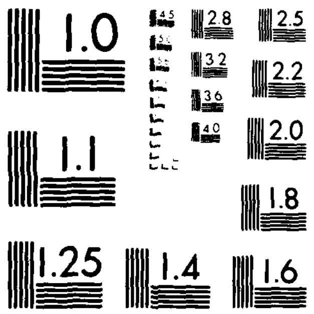
The logic for decoding instructions is one of the most vital functions in the implementation of a universal emulator. Decode must be handled by a series of firmware instructions rather than specific decoding logic. Since the firmware speed cannot match the hardware speed in the target system it is important that efficient hardware support the firmware. In the IEU, there should exist two levels of support. A special series of multiplexers are required to allow the choice of any arbitrary eight bit field from an instruction followed by a mask word which allows the choice

AD-A086 065 HARRIS CORP MELBOURNE FL GOVERNMENT COMMUNICATION SY--ETC F/G 9/2
MULTIPLE MICROPROCESSOR SYSTEM (MMS) DESIGN STUDY. VOLUME I.(U)
MAR 80 F30602-78-C-0114
UNCLASSIFIED RADC-TR-80-33-VOL-1 NL

2 of 2

AD A086065

END
DATE FILMED
8-80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5

of any combination of bits in the 8 bit field. After a field is selected, it may be passed directly to the sequencer for a multi-way branch into microcode or to the mapping memory for lookup of the microcode address. Additionally, a relocation constant is used to locate the position of the lookup table in the mapping memory or determine the starting address in the multi-way branch. The instruction decode logic is not limited for use in decoding instructions. The selection and lookup logic may be used for I/O address, condition code, and other general translations. See Figure 3.2.1.2.

3.2.1.3 Conditions Codes

The conditions codes within the IEU are basically of two types. The first set are the loadable conditions which exist in a special purpose condition code register which is loadable and readable by the user. Load condition codes are completely definable by the user. They may be individually set or reset by the user and are expected to be used to represent target machine operating modes or any esoteric set of 16 conditions the user may devise. The second type are the ALU condition codes which are a set of 16 MMS supplied conditions generated by the hardware ALU. The user has the power to load or reload at any time the entire set of codes. However, the meaning of these codes are specific and fixed. In order to output target system status work, it will be necessary to translate the MMS codes into his target system defined order. This action will ordinarily be infrequent and can probably be handled in one, two, or three micro cycles. See Figure 3.2.1.3.

FIGURE 3.2.1.2 INSTRUCTION DECODE LOGIC

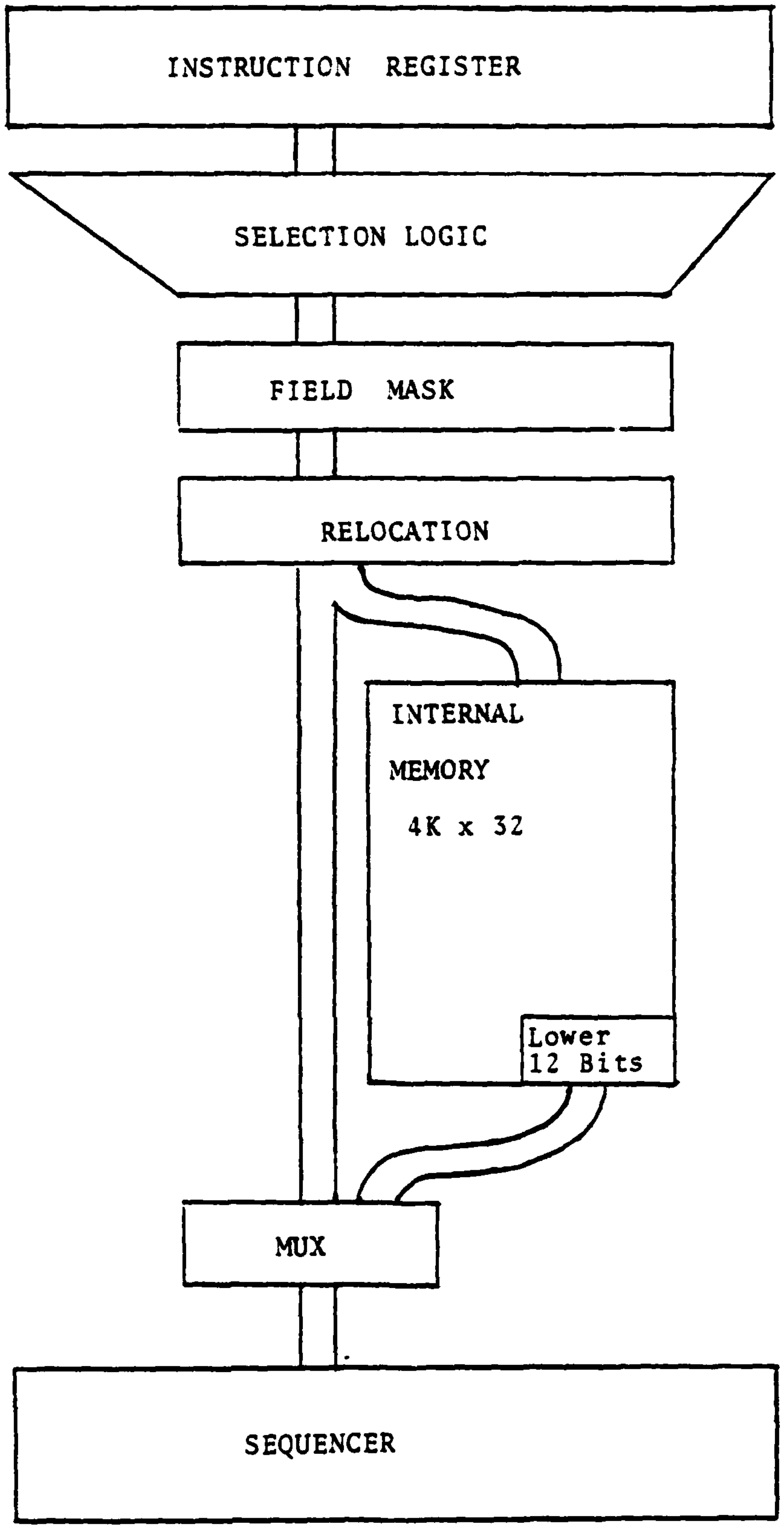
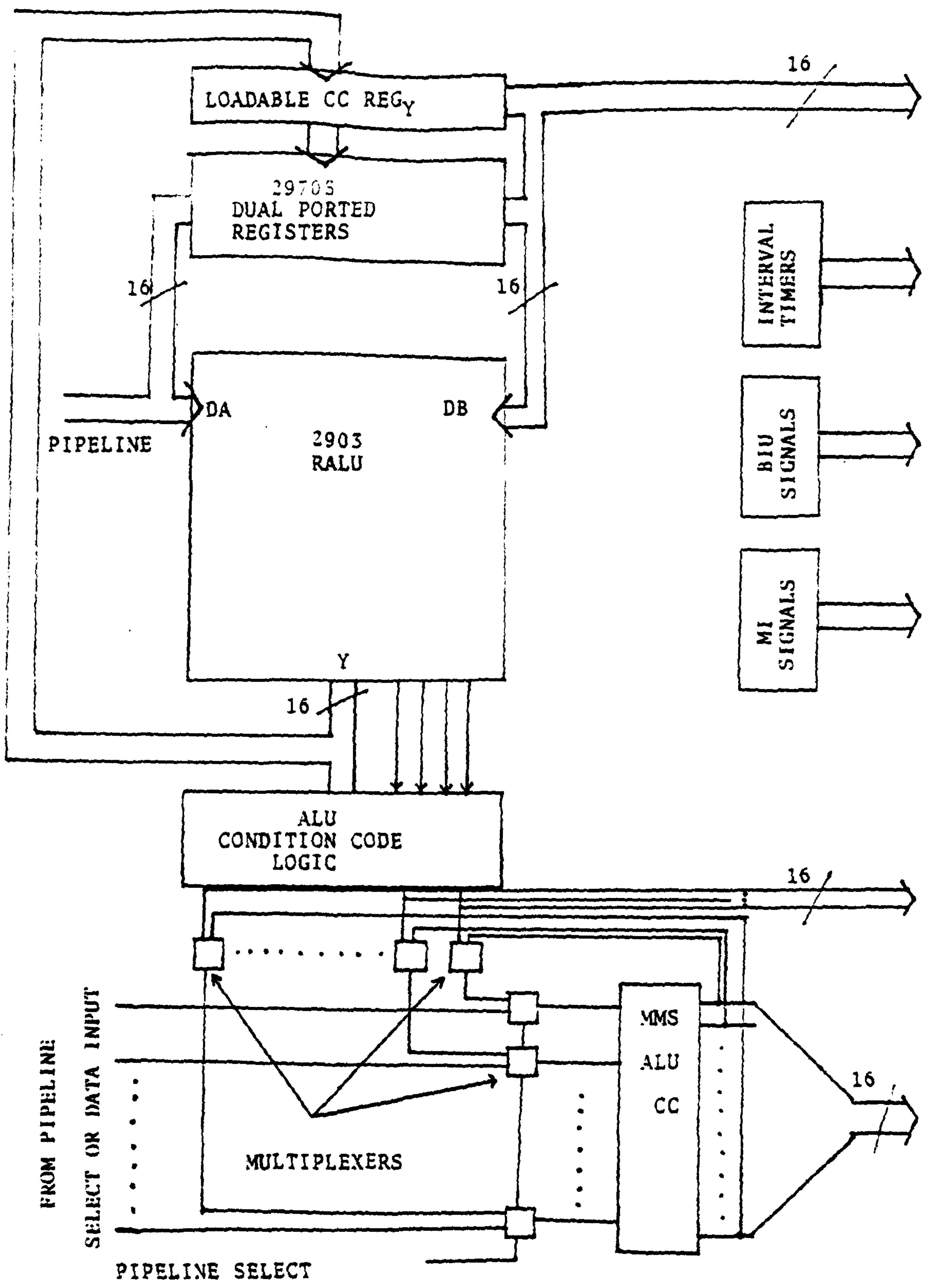


FIGURE 3.2.1.3 CONDITION CODE LOGIC



3.2.1.4 Arithmetic Logic Unit

The arithmetic logic unit will also be of the 2900 series architecture. The ALU shall have the capability to perform the add, shift, and restoration of the answer within 200ns. The ALU shall be sixteen bits in width. Additionally, there shall be a special hardware chip capable of sixteen by sixteen binary ot two's complement multiplication. There will be no special purpose hardware present to handle floating point arithmetic due to the wide variety of floating point formats. An additional bank of 16 dual ported registers will be utilized by the ALU. Their purpose will be to provide fast context switching between user and executive modes. The intended use of the registers within the ALU will be as scratch pad and temporary registers for intermediate calculations within instructions. They may also be used for any target system register which will not be monitored by the PMS function. Of course, the actual usage of these registers will be totally subject to the user's microcode.

3.2.1.5 Internal Memory

The internal memory (IM) of the IEU is approximately 4K X 32 in size. In the lower order 16 bits, the IM holds a variety of different values, including target system registers, I/O device emulation registers, map tables for mapping op codes and I/O addresses, and translation tables for condition codes. The internal memory is the major piece of hardware within the IEU for performance monitoring. In the upper 16 bits of the internal memory exists PMS flags and event codes that are associated with the particular element stored in the lower sixteen bits. Whenever the particular location in the internal memory is accessed, a bit from one of the upper positions will cause an automatic trap to a specified location in the PE executive which will process the PMS event and return control to the executing program. The choice of storing user registers inside the internal memory is convenient due to the already existing decoding selection logic at the input of the internal memory. In this way a register contents is available at the internal memory's output. This particular organization of internal memory yields an extremely powerful performance monitoring system which can monitor as much of the target system as the user desires. Of course, the user must spend time setting the appropriate flags and codes to monitor each individual item. A supplemental explanation of PMS is presented in Appendix A.

3.2.1.6 Local Time Alignment Control

Time Alignment control within the IEU consists of a series of registers and associated logic. There is a local pseudo time register which is loaded from microcode by the user at the appropriate times, and a master pseudo time register which is loaded during setup time by the facilities control processor. At the heart of the logic is an adder/subtractor which adds the value in the master pseudo time register to an accumulator upon signal from the master TAC and subtracts the value in the local pseudo time register from the accumulator upon signal from the IEU. Additionally, there is a window register which holds the value of half the window size. This value is periodically compared to the differential value in the accumulator. When the accumulator value is more positive than the window register value , a stop signal is sent ot the master TAC. When the accumulator value is more negative than the negative value at the window register, an idle signal is sent to the instruction execution unit.

3.2.1.7 Communication Network

Within the IEU, there is a massive amount of communications that must take place among the registers and other devices present. The exact paths between all devices in the IEU are contingent to a large extent on the format chosen for the microcode word.

Basically, it is known that communication will be implemented in three ways, First of all, every writable device will have an access path to a central location or bus. This will be necesssary due to the requirement that the entire system is configurable and loadable from the facilities control processor. Also, having a common bus will allow micro-debug to be carried out by a program executing in the microstore which will transfer the contents of each register to a special debug port connected on the bus. The BIU will be responsible for transfering the contents of this debug port to the system bus where the facilities control processor may gain access. Secondly, there will be a small set of dedicated busses within the IEU that handle communication from specific fields in the microcode word. The third type, of course, is the single dedicated control lines that go to specific elements of hardware such as multiplexer select lines, register load and device output enable signals.

3.2.1.8 External Interfaces

External interfaces refers to the hardware required for communication between the IEU and devices external to the IEU, but within the processing element. These interfaces are mainly concerned with IEU communication with the memory interface and the bus interface unit. Specific signals to begin translation, increment address, and access memory must be sent to the M1, while clock and reset signals must be sent to the BIU. Each interface has specified registers through which transfer of address and data are performed. The IEU is responsible for monitoring the status of these devices as well. The MI presents status signals of busy, valid translation complete, address not found, I/O address found, block monitored by PMS, and shared resource request being made to the IEU. The BIU presents service request and busy status signals to the IEU.

3.2.1.9 Input/Output Support Hardware

The biggest area of I/O support comes under the auspices of firmware and PEXE. However, there are two specific areas in which the hardware helps. One is the addition of a small bank of interval timers. These timers are utilized in distributed I/O emulation. The interval times are initiated by specific device handler routines to a certain value of pseudo time which the handler desires to elapse before it regains control. When a particular interval timer reaches zero, an MMS interrupt is logged which eventually leads to the servicing of a particular I/O handler routine. The other type of special hardware to be added is some manner of priority encoding. This is needed to order the priority of MMS interrupts coming in and provide an entry point into the PE executive.

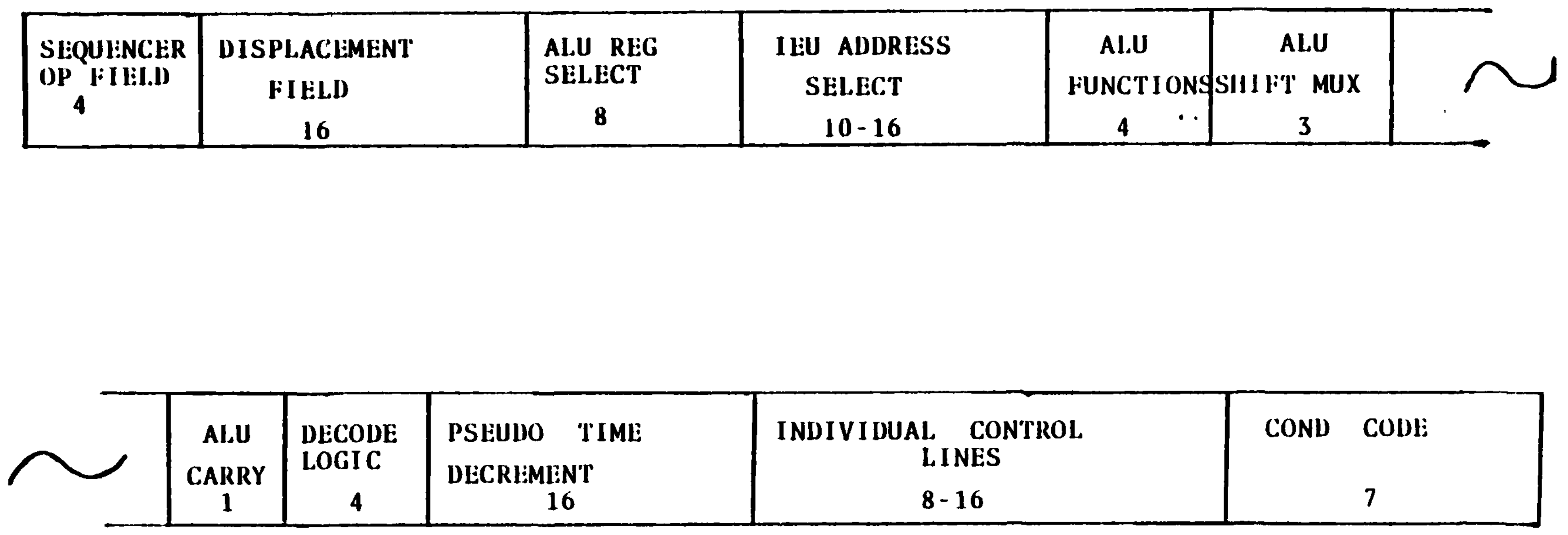
3.2.1.10 Microstore and Pipeline Register

The microstore and associated pipeline register in the IEU will be approximately 96 bits in width. The control store itself will be high speed RAM approximately 4K X 96 bits in size. A complete breakdown of the microword will be presented in the next section.

3.2.2 Microcode Word

The format of the microcode word is closely associated with the specific hardware design. Depending upon how judiciously chosen, it can greatly help or impair the efficiency of the microprogram. A first pass look at the microcode word is shown in Figure 3.2.2, with an outline explanation of the fields presented in the 2 tables which follow Figure 3.2.2. One possibility for a change is the creation of a separate field for the decode mask word due to the possibility of arithmetic and condition code logic being utilized on the same cycle as selection of a register within the internal memory. Other possibilities involve the creation of a separate field for the second ALU register select, decrease of ALU register select field to 5 ro 6 bits, decreasing the condition code input select to 6 bits due to the elimination of the 32 PMS condition bits, overlap of some currently separated fields, encoding of some currently decoded fields, or decoding some currently encoded fields. It is now estimated that the combination of ALU carry, function select, and shift mux select will take anywhere from 11 to 14 bits, depending upon the ALU device used and the flexibility required by the user.

MICROWORD FORMAT



TOTAL NUMBER OF BITS: 80-96

Figure 3.2.2

MICROWORD FORMAT

	NUMBER OF BITS
SEQUENCER OP CODE FIELD	4
DISPLACEMENT FIELD	16
A. NEXT ADDRESS (MICRO)	
B. ALU COND CODE SELECT	
C. LOADING OF REGISTER	
1. ALU COND CODES	
2. LOADABLE COND CODES	
3. MASK WORD FOR DECODE	
D. ALU REG SELECT FIELD	
PSEUDO TIME DECREMENT FIELD	16
ALU REG SELECT (SRC/DEST)	8
IEU ADDR SELECT (M-INST TYPE)	10-16
ALU CARRY IN	1
ALU FUNCTION SELECT	4
ALU SHIFT MUX SELECT	3
DECODE LOGIC SELECT	4
INDIVIDUAL CONTROL LINES	8-16
A. PT DECR	
B. MI START SIGNAL	
C. MI INCR SIGNAL	
D. MEMORY BYTE/WORD SELECT	
E. BIU CLOCK SIGNAL	
F. BIU RESET SIGNAL	

COND CODE INPUT SELECT 7

A. 16 ALU CONDITIONS

B. 16 LOADABLE CONDITIONS

C. 32 PMS CONDITIONS

D. 16 INTERVAL TIMER INTERRUPTS

E. 5 MI CONTROL

   1. COMPLETE-CORRECT

   2. ERROR

   3. PMS BLOCK

   4. SRC ACTION

   5. I/O ADDRESS

F. 2 BIU

   1. SERVICE REQUEST

   2. BUFFER FULL

G. COMBINATIONS

79-95 BITS WIDE

3.2.3 Memory Interface

The responsibilities of the memory interface are to examine, translate, and send out memory addresses, as well as present that examination and the status of its actions to the IEU. A diagram of the memory interface is in Figure 3.2.3, for reference. Discussion of the device will be outlined in the same manner that the blocks in the diagram are layed out. Portions of the IEU and BIU interfaces have already been presented in previous discussion in the IEU section.

3.2.3.1 Memory Interface Sort Table

The sort table, along with the sequencing logic and comparison logic, actually performs the functions of a sophisticated *content addressable memory* with the added feature of translation of the input address ot physical MMS address. Research done of any specific chips to implement a content addressable function has yielded only two chips both with total of 16 bits which are clearly inadequate.

The configuration of the sort table will be approximately 2K words by 64 bits wide. Entries in the table will be in the form of 2 words of 64 bits each. In the first word the beginning and ending addresses of a target system block of memory will be stored as two 32 bit fields. In the second word the first 32 bits will be used as a relocation constant for translation of a target system address to physical address. The last half of the word will be descriptor bits and control signals. They will include bits that specify the memory block to be emulated shared or local, monitored

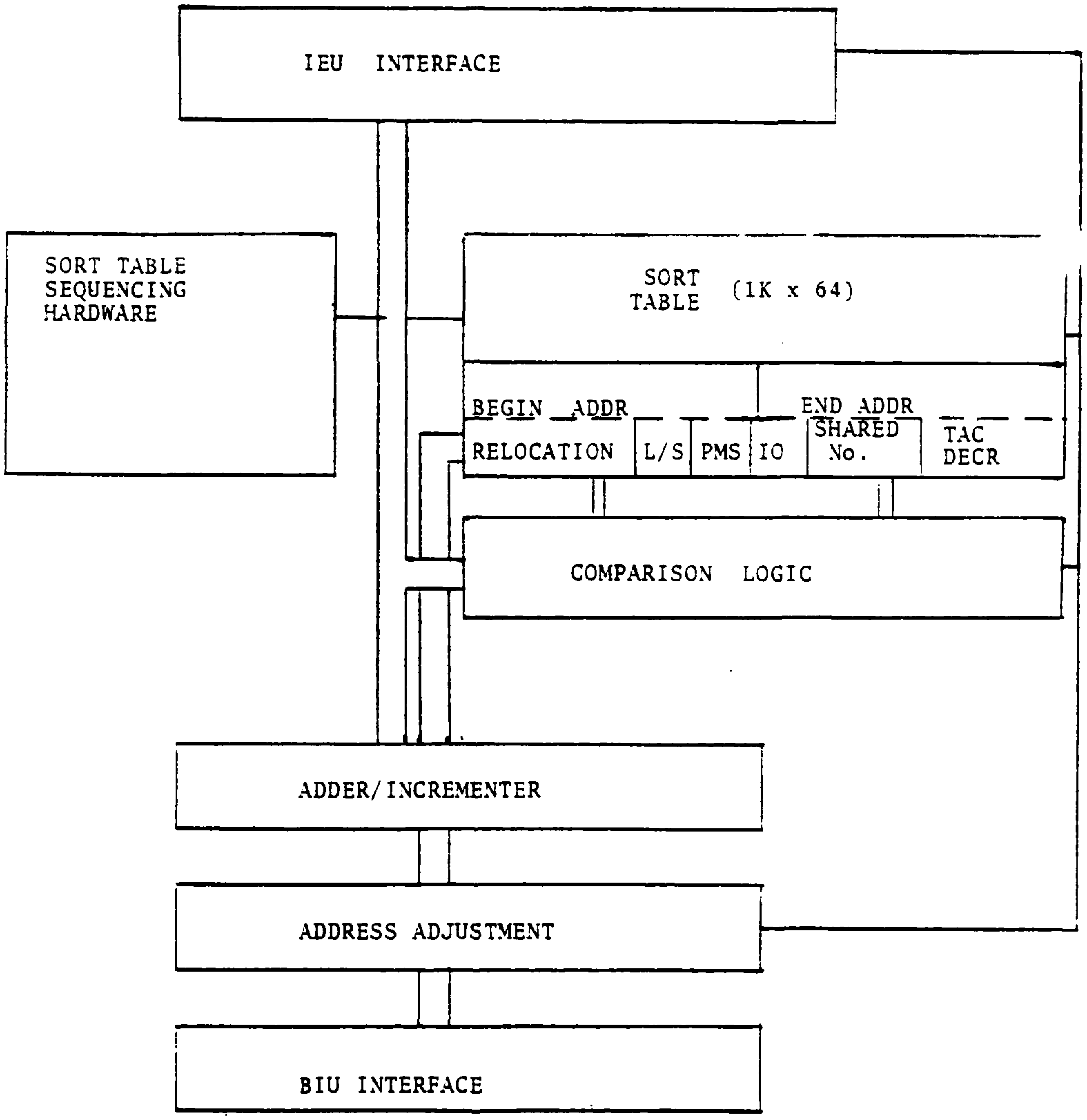

IEU INTERFACE
SORT TABLE SEQUENCING HARDWARE
SORT TABLE (1K x 64)
BEGIN ADDR
END ADDR
RELOCATION
L/S
PMS
IO
SHARED No.
TAC DECR
COMPARISON LOGIC
ADDER/INCREMENTER
ADDRESS ADJUSTMENT
BIU INTERFACE

FIGURE 3.2.3

by PMS, memory mapped input/output as well as control to sequencing logic bits. Particular fields which will be required are a TAC decrement field that specified the cycle speed of the emulated memory, a PMS field that specifies flags and the event number associated with accessing this particular memory block, and a shared block number which is sent to the shared resource controller to specify the shared block to be accessed.

3.2.3.2 Sort Table Sequencing Hardware

The sort table will need some sequencing logic to be able to step through the comparisons needed to find the block in which the memory address exists. The hardware will be able to sequence through even addresses (those which contain the first word of an entry) and be able to remember the last block in which the memory address was found. This will allow for a more efficient search for the next memory address presented to the memory interface.

It is possible that two or three registers may be used to hold the most recently used addresses, but it is viewed as overkill to attempt to implement a complex sorting algorithm. Logic used here is viewed to be fast MSI/SSI schottky logic.

3.2.3.3 Comparison Logic

The comparison logic is utilized to determine whether or not the presented address is within that particular block memory. The logic will consist of two 32 bit comparators. One compares the beginning address to the input address while the other is comparing the input address to the ending address of the block. If the input address is found to exist between these addresses, signal is sent to the sequencing hardware to begin processing on the next word in the entry.

3.2.3.4 Adder/Incrementer

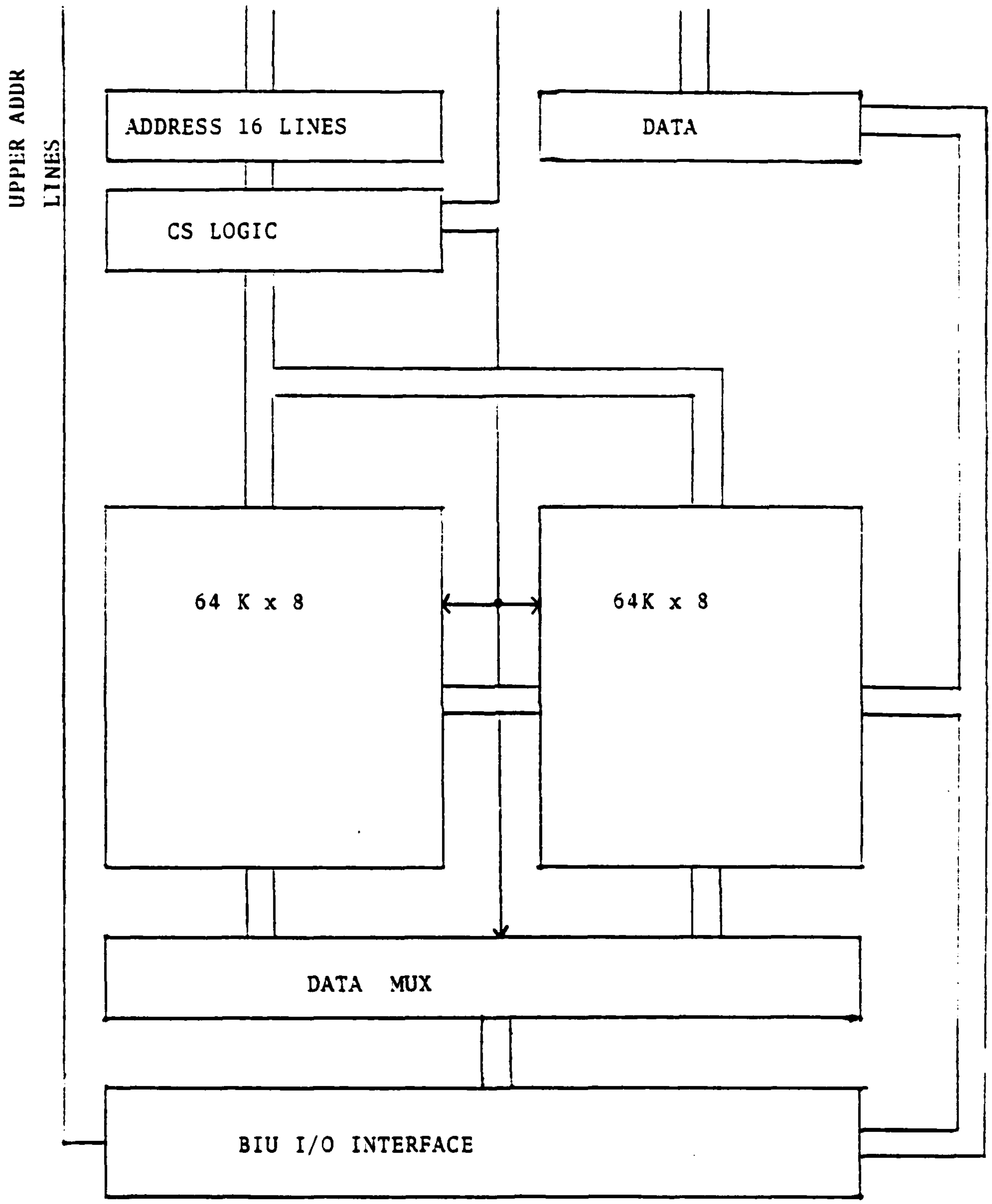
A 32 bit full adder will exist primarily to add the appropriate relocation constant to the address input. This is basic in the translation of the target system address to the MMS system address. The adder is used as an incrementer in the case that requires two or more bus cycles to implement the target system memory access.

3.2.3.5 Address Adjustment

The relocation constant placed in the sort table is chosen so that the address is relocated with respect to the type of target system address, i.e., 8 bit, 16 bit, 32 bit, 64 bit data elements. Thus, what is produced by the relocation is a relocated address, but not necessarily a byte address which must be sent out over the MMS address lines. There is no way that a single constant can be chosen for a block of consecutive address that will relocate and translate to a byte address. Therefore, the address adjustment logic is present to shift 16, 32, and 64 bit addresses left one, two, or four times, respectively, to form an MMS byte address.

3.2.4 Local Memory Module

Refer to Figure 3.2.4 for a block diagram of the local memory module. Central to the module are the memory blocks themselves. They will be 200ns to 450ns cycle time RAM's arranged in byte wide modules. The number and size of the modules is dependent upon the width of the data of the system bus. If 16 bits wide, there will be two modules of 64K words each. If the bus is 32 bits wide, there will be four modules of 32K words each. In either case,the total size of the memory will be one megabit. The



LOCAL MEMORY FIGURE 3.2.4

bus interface logic will be responsible for decoding the upper 16 bits of the 32 bits of address present on the bus, and only the lower 16 bits will need to be sent to the memory module itself. The internal chip select logic will be used to select which banks of memory should be activated for the current access. There will be a data multiplexer network present to shift the data into the proper banks, as well as outputting the data right justified on the data portion of the bus. There will be specific interface logic to handle memory read operations. Received in the data portion during the read request will be the code of the processing element making the request. After the access to memory has been made, it is necessary for this logic to format the read reply message.

4.0 TIME ALIGNMENT OF THE PE's IN MMS

The concept of time aligning the PE's in MMS is performed by using a time window and a time reference called Master Pseudo-Time (MPT).

During run time, any PE's which are within the time frame of the window may continue execution. Any PE which falls behind the window must halt the advancement of MPT until the PE can execute long enough to move back into the window. Any PE which moves ahead of the window must enter an idle state until MPT advances enough for the PE to move back into the window.

4.1 <u>Implementing Time Alignment of the PE's in MMS</u>

The time alignment function of MMS is performed using the following elements:

1) A master time alignment controller, TAC on the central communications segment.*
2) A local pseudo-time accumulator found in each PE.
3) Two "wired-or" lines which are part of SBS and are connected to all PE's in MMS.

The two "wired-or" lines are the time increment lines, TIC, and the stop master pseudo-time line, SPT. The TIC line allows the TAC to send TICS to each PE on SBS. Each TIC indicates a pseudo-time increment has occurred. The wired-or SPT line allows any PE which falls behind MPT to temporarily halt the TAC from issuing any TICS until the PE can advance in pseudo-time and realign itself with the other PE's. Once realigned the PE releases the SPT line, which allows the TAC to continue.

* Also referred to as the master segment.

4.2 The Time Alignment Controller

The TAC can issue TICS to the PE's in any one of the following four modes:

Mode 1: Free run at maximum rate - continuously issue TICS unless a stop is sent via the SPT line.

Mode 2: Free run at slow rate - continuously issue TICS unless a stop is sent via the SPT line.

Mode 3: Burst Mode - issue a specified number of TICS and stop.

Mode 4: Single step - issue a single TIC on command from the FCP.

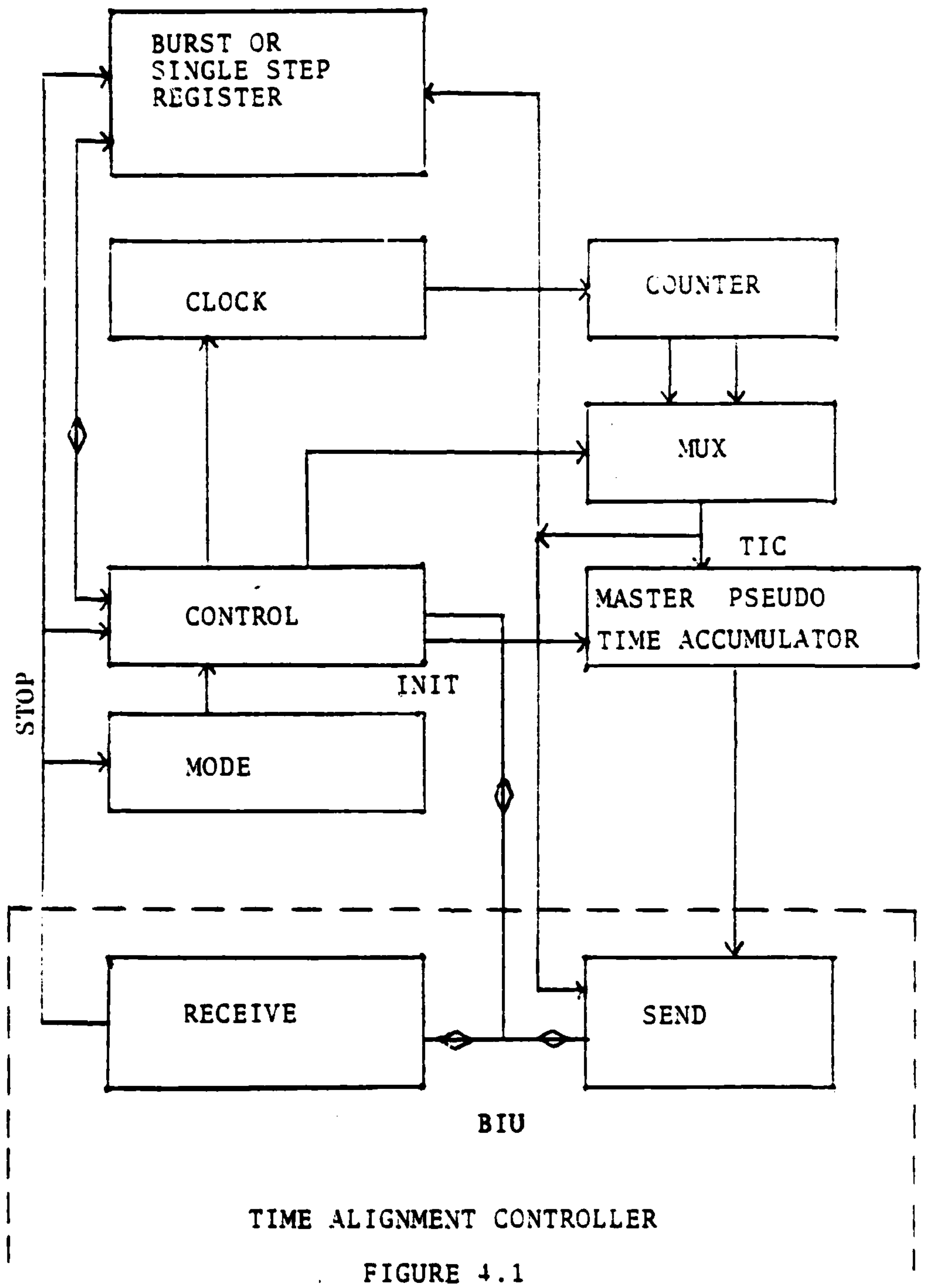
A functional block diagram of the TAC is shown in Figure 4.1. The TAC maintains MPT by accumulating every TIC issued with a master pseudo-time accumulator. This accumulator and the associated control registers are accessible by any other devices on SBS.

4.3 The Local Pseudo-Time Accumulator

The local pseudo-time accumulator, LPA, can operate in either of the following modes:

Mode 1: Time alignment mode - a PE uses the LPA to time align itself with the other PE's in MMS.

Mode 2: Bypass Mode - the LPA can be bypassed, so that, a PE can operate in free run mode.



TIME ALIGNMENT CONTROLLER

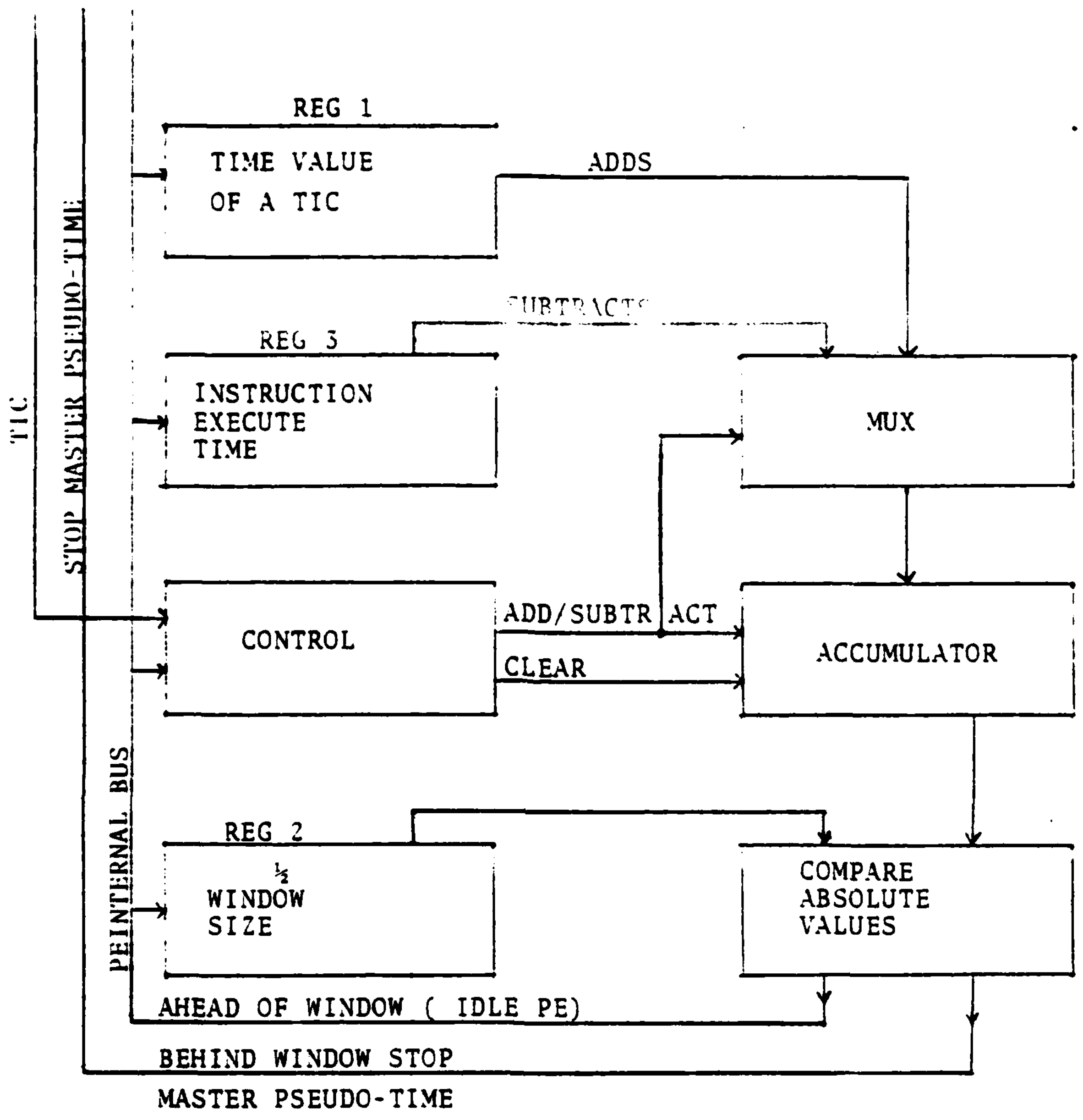
FIGURE 4.1

A functional block diagram of the LPA is shown in Figure 4.2. The LPA consists of 3 registers and a differential accumulator. The first register is loaded during system initialization and is used by the operator to define the value of a TIC, such as, 10ns. The second register is also loaded during system initialization and is used by the operator to define the time frame of the pseudo-time window, such as 500ns. The third register is loaded by the PE during the execution of each macro instruction and is used to store the amount of pseudo-time required to execute a specific macro instruction.

The differential accumulator is used to calculate the difference between the amount of MPT issued by the TAC and the amount of pseudo-time used by the PE during the execution of a macro instruction.

The LPA operates in the following manner:

1) The value stored in register 1 is added to the accumulator every time a TIC is issued by the TAC.
2) During execution of every macro instruction the valued loaded into register 3 by the PE is subtracted from the accumulator.
3) The differential value in the accumulator is compared ot the time window stored in register 3. This comparison results in a flag being set if the PE is ahead of pseudo-time or if the PE is behind pseudo-time.



LOCAL PSEUDO-TIME

ACCUMULATOR

FIGURE 4.2

4) At the completion of executing a macro instruction the PE can test these flags. If either flag is set the PE must either enter a idle state and continue execution, respective to the states mentioned in part 3.

4.4 The Effects of Time Alignment

Time alignment effects the PE's by forcing them to continuously start and stop as they move in and out of the pseudo-time window. This start/stop action is controlled by the combined effects of the pseudo-time window size, TIC value and the amount of pseudo-time required to execute a macro instruction.

The window size can affect how long a PE remains in the time window and can continue executing macro instructions. A large time window provides the PE's with a large enough time frame to execute several macro instructions. A small time window tends to make PE's single step. For a small window most PE's will move ahead of the window after executing each macro instruction. When this occurs the PE's must idle until pseudo-time advances.

The TIC size affects the rate at which the time window advances in pseudo-time. A large TIC size has the effect of jumping the time window forward in pseudo-time. This results in PE's which are in the window or ahead of the window falling behind the window and halting the TAC. A small TIC size has the effect of allowing the PE's to almost always be ahead of the window. As the window slowly advances in pseudo-time a PE will fall into the window, execute a single macro instruction and move ahead of the window again.

The amount of pseudo-time required to execute a macro instruction also affects how a PE moves with respect to time window. An instruction which requires a large amount of pseudo-time can jump a PE far ahead of the time window. An instruction which requires a small amount of pseudo-time can drop a PE behind the time window.

The combined effects of the PE's either jumping far ahead or dropping far behind the window or the window jumping far ahead of the PE's requires the LPA to have a bit size large enough to prevent and overflow or underflow condition from occurring.

When the units of the pseudo-time TICS are in nanoseconds, a 32 bit LPA can maintain a differential value between master pseudo-time and PE pseudo-time of approximately $\pm$ 4 seconds. With the window size defined to be in the microsecond and millisecond range, the LPA should be able to accomodate any macro instruction which requires a long period of pseudo-time to be executed. An example of such an instruction is a Z-80 CPU block move instruction. To move 64K bytes of memory with this instruction using a 2.5MHz CPU requires approximately 0.5 seconds.

5.0 I/O EMULATION CONCEPTUAL DESIGN

5.1 Overview

Emulation of I/O devices is one of the more difficult tasks to be accomplished within the MMS primarily because emulated I/O data streams must flow and be controlled in a wide variety of ways simultaneously. Some I/O data streams are completely contained within a PE, some within the emulation engine, other data streams are "sourced" or "sinked" within the facilities control or I/O processor, while some data streams may originate external to MMS.

An objective of the I/O emulation conceptual design presented here is to provide a control and communications structure for the MMS such that user action is required only for emulation of I/O devices themselves and not for the more mundane tasks of file handling and communications idiosyncrasies. In addition, a control structure is provided so that I/O emulation can be specified at a high level through the system generation language and environmental command language. This high level specification is applicable once the microcode modules that perform I/O device emulation are complete.

A complete discussion of I/O emulation encompasses the following topics:

1. System Generation Language Support
2. Environmental Command Language
3. I/O Processor (IOP) Software Support
4. IOP Interface
5. PE Executive (PEXE) Support
6. I/O Device Emulation Code

The first three topics deal with the systems software aspects of I/O emulation and is detailed in the software conceptual design rather than in this section. However, an outline of objectives for this software is included below. The last three topics are detailed in the discussion below.

5.2 Systems Software Objectives

The major objectives of MMS system software relative to I/O emulation is to provide a well structured organization that makes previously written I/O device emulation modules easy to configure and use in a target system emulation. The user being addressed here, is the target system user as opposed to the microcode or emulation writer.

As noted previously in the PE software discussion, a distinct separation is maintained between instruction emulation code modules and individual emulation code modules. This separation or segmentation into modules is enforced to gain flexibility within emulation. Without this segmentation, the target system user would be frequently forced into modifying emulation code manually to meet particular requirements, since the emulation code would be very target system dependent. With the segmentation of microcode code into modules, the emulation code becomes target system independent and thus can be captured or reused over many similar target system emulations without modification. In addition to modularization of emulation code, parameterization of the code is a desirable feature. It allows high level specification of common target system variables such as baud rate or device codes.

The system software provides the structure for implementation of both the modularization and parameterization of user code. It provides the high level specification languages (i.e. system generation language and environmental command language), library facilities, linkage facilities for emulation code, and run time routines for the I/O processor.

5.3 IOP Interface

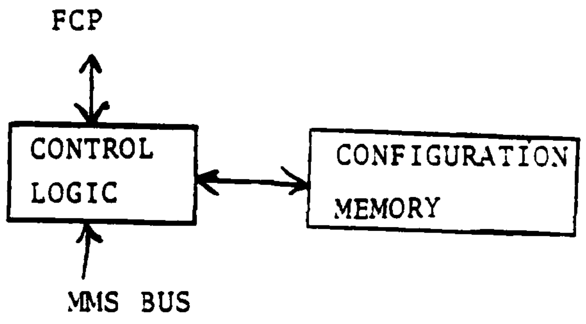
The IOP interface interconnects the I/O Processor to the emulation engine by way of the MMS bus. Its purpose is to perform those I/O emulation tasks associated with communication between the IOP and the emulation engine, and other identified tasks requiring hardware support. There are three main components within the IOP interface.

- IOP Ports
- IPC Buffer
- Interval Timers (Pseudo-Time)

Each of these are discussed below.

5.3.1 IOP Ports

The IOP Ports are the largest component of the IOP interface. They perform the major communication function between the emulation engine and the IOP. It consists of an interface between the MMS bus and the FCP and constitutes effectively 2000 configurable ports and some memory that contains the configuration data as depicted below.



Each of the IOP Ports is unidirectional, configurable in direction (input or output) and in width (1, 2, 4, or 8 bytes). The most distinguishing feature of the IOP Ports, however, is the protocol translation that they implement.

From the FCP side, an IOP port is configurable into either a programmed I/O mode (interrupt) or a Direct Memory Access (DMA) mode.

From the MMS side, the IOP ports support the MMS bus protocol. All transfers are accomplished in a programmed I/O fashion without interrupts. That is, all transfers are under explicit PE control. What is unique about the MMS side of the IOP Port is that all handshaking of data into or out of port is completely transparant to the emulation code and provides automatic pseudo-time synchronization within the emulation engine. Consider a read request as an example. When a PE issues a read request the IOP Port will normally fetch prestored data from the port configuration memory or from the FCP, and returns the data to the requesting PE. However, if that data is not immediately available the IOP Port simply waits until the FCP clears the situation. The requesting PE in this situation also enters a wait mode eventually falling behind the master pseudo time window and halting (thus synchronizing)the entire emulation engine. To the PE microcode then, an IOP Port appears to be like any read/write memory location, thus simplifying I/O emulation code development.

5.3.2 IPC Buffer

The IPC Buffer is similar to the IOP Ports in that it provides a communication path between the IOP and the emulation engine. The distinction is the method whereby the communication is accomplished. In the IOP Port data sent from the IOP to a PE is held or stored in the port (the interface) until the PE requests the data. The IPC buffer provides a mechanism where by data is sent directly to the PE in the form of an IPC (Interprocessor Communication) message. This mechanism is identical to that used by PE's in communicating directly between each other. Note that the IPC buffer is utilized only in the direction from IOP to PE. In the opposite direction an IOP Port would be utilized.

5.3.3 Interval Timers

A bank of 64 timers with master pseudo-time as an input is included in the IOP interface so that the IOP can effectively emulate I/O devices or I/O data streams with proper pseudo-time synchronization to the emulation engine. The timers are utilized by the IOP in much the same manner as a similar bank of interval timers within each PE is used. The major difference associated with the IOP interface timers is that they include a provision for stopping master pseudo-time when a time out occurs. This provision is necessary since the IOP is not directly connected into the pseudo-time synchronization mechanism like a PE.

5.4 PE Executive Support

The PE executive (PEXE) provides common support functions within a PE for I/O emulation purposes. It provides; interrupt routing for IPC's and the pseudo interval timers; a set of software implemented pseudo interval timers; DMA scheduling; and macro interrupt scheduling. All of these functions have been detailed in the PE software discussion.

5.5 I/O Emulation Code Examples

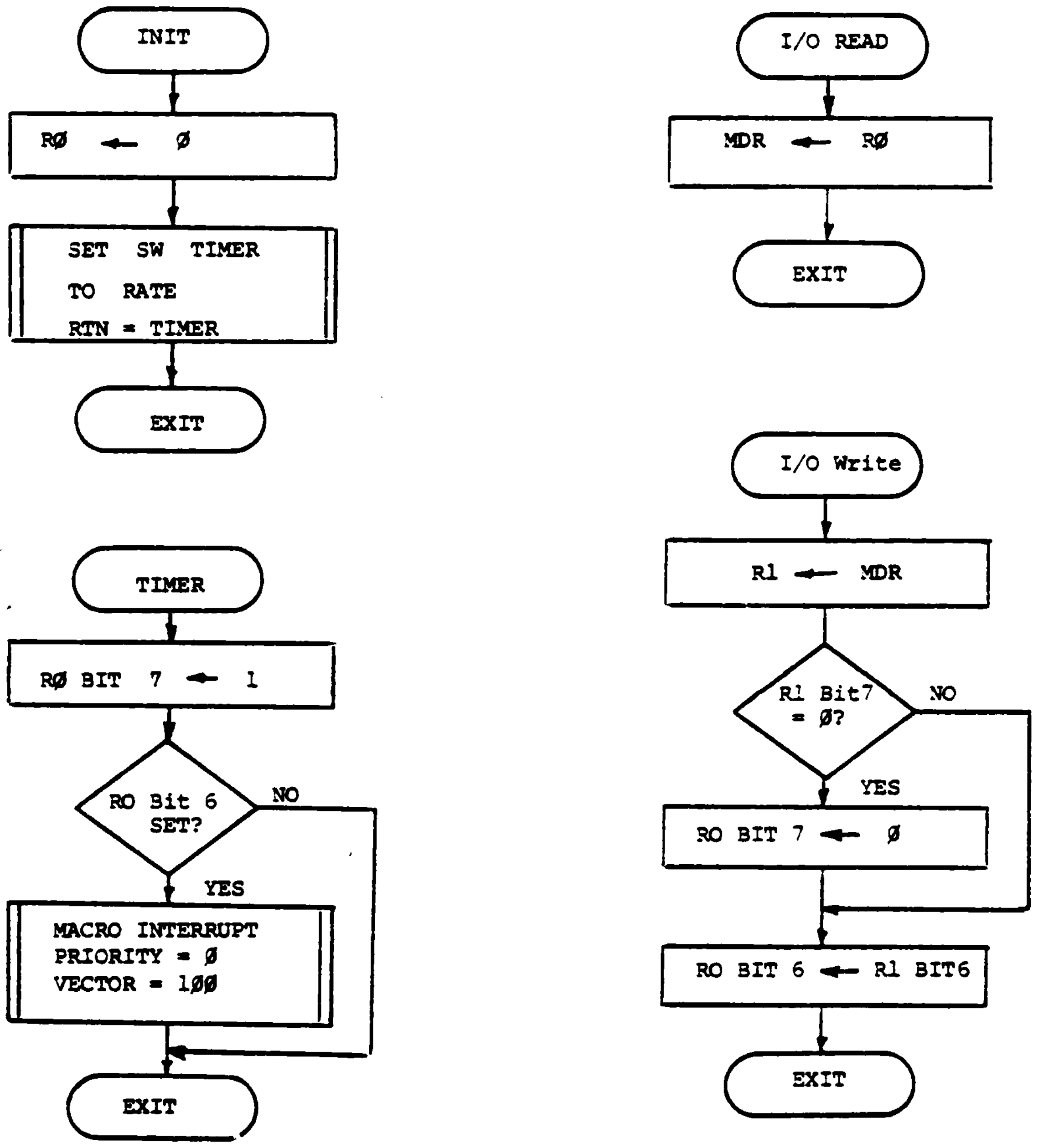
Two examples of I/O emulation code are presented here to illustrate the concepts being strived for. The first example is a very simple I/O device, a real time clock for an LSI-11. It demonstrates the concepts of modularity and parameterization of I/O code. The second example, an RS-232 interface, demonstrates the use of the IPC capability of the MMS bus. More importantly, though it demonstrates the commonality of code that may, in various target system emulations, be interconnected to other processors or to the external environment.

5.5.1 Real Time Clock Example

This example, the KPV11 line frequency clock for the DEC LSI-11 processor, is a simple device emulation that illustrates modularity and parameterization of I/O emulation code. To the target system code, this device appears as a single address with two status bits. One bit enables interrupts and the other is a monitor bit set by the device at a periodic rate and cleared by the processor. Since the device has limited configurability, the emulation code has only two parameters which can be specified by the system generation language. They are:

- o RATE - 50 or 60 Hz (Default 60Hz)
- o Device Code - (Default 177546 HEX)

Emulation code for this device has four entry points. Flow charts for the emulation code are given in Figure 5.5.1. The initialization routine is entered upon a system bit and performs a service call to the PE executive (PEXE) to establish a software timer. When that timer overflow occurs, the



NOTE: ▭ = PEXE SERVICE CALL

FIGURE 5.5.1 Flow Chart for real time clock.

timer routine is executed. It sets the monitor bit and then generates an interrupt, via a PEXE service call, if the interrupt enable is set. A priority and a vector address are passed to the PEXE service routine to be used by a macro interrupt routine in the instruction emulation code. The two other routines, I/O read and I/O write, are entered when the instruction emulation code performs an access to the device address. The decision branch in the I/O write routine implements the fact that the monitor bit can be only cleared by the target system code.

5.5.2 RS232 Interface Example

This example of an asynchronous line interface demonstrates the emulation of a more complicated device than the previous example, and also shows the use of the IPC feature of the MMS bus. The targeted device, a DLV11 serial line unit for an LSI-11, presents four registers to the target system software:

- o RCSR - Receiver control and status register
  One bit for interrupt enable one bit ready flag.
- o RBUF - Receiver buffer register
  Eight bits of data
- o EXSR - Transmit control and status register ready and interrupt enable bits.
- o XBUF - Transmit buffer register eight data bits

Because of the additional complexity of this device eight parameters are available for high level specification. They are:

- o Rate in characters per second
- o Device Code
- o Transmit vector address

- Receive vector address
- Transmit interrupt priority
- Receive interrupt priority
- Transmit destination

All of these parameters are self-explanatory except perhaps for the last one. Transmit destination will be specified during system generation with a processor and device number or with an IOP port number. This information would be translated by the system generation software into an MMS bus address. In using this bus address to send the data, the transmitting PE utilizes the IPC message capability of the MMS bus. Note that the same mechanism (and same code) is used for communicating with another PE in the emulation engine and with an external destination through the IOP interface.

The emulation code for this device consists of an initialization routine, four routines for the transmit function as flow charted in Figure 5.5.2-1, and four routines for the receive function as flow charted in Figure 5.5.2-2. The PE registers used to emulate the device registers are:

o R∅ - RCSR

o R1 - RBUF

o R2 - XCSR

o R3 - XBUF

Note that a software timer is used in the transmitter to maintain pseudo-timing. No timer is needed in the receiver since pseudo-timing is provided by the originating transmitter.

The emulation code routines presented in these examples are particularly simple to construct because of the communications and software support provided by MMS. This simplicity contributes greatly to the ease of use feature of MMS.

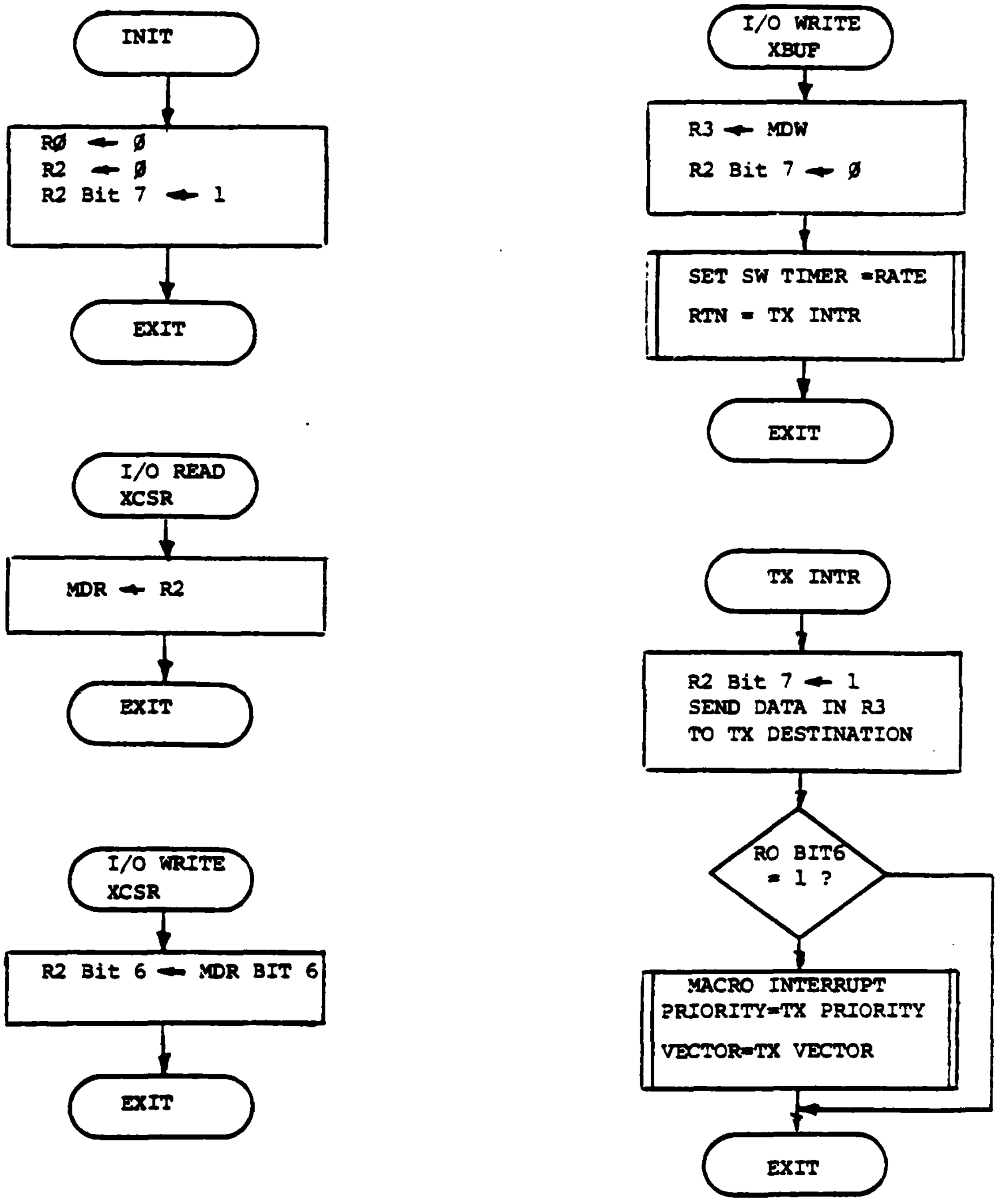


FIGURE 5.5.2-1 FLOW CHARTS FOR TRANSMIT FUNCTION.

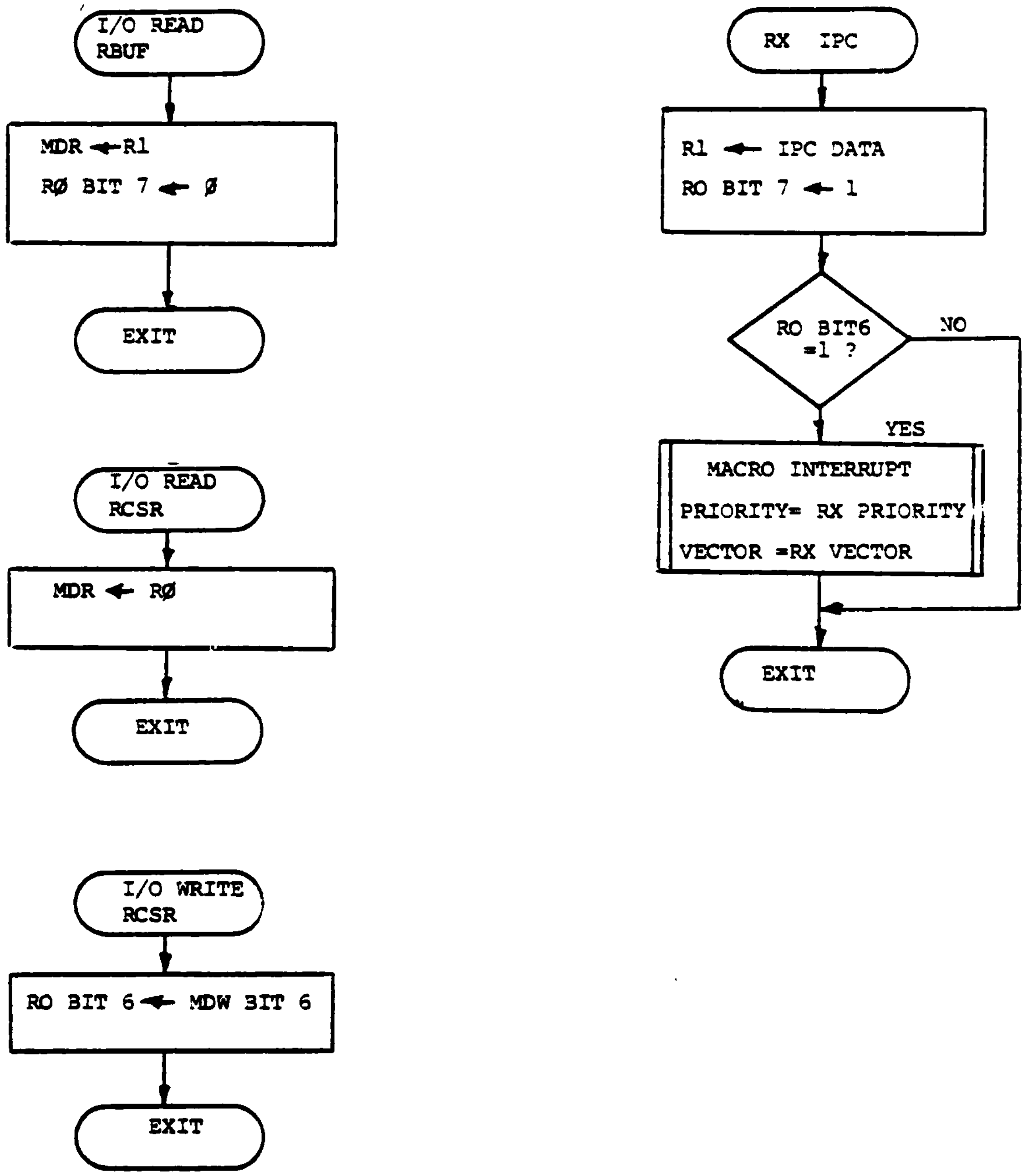


FIGURE 5.5.2-2 FLOW CHARTS FOR RECEIVE FUNCTIONS

6.0 SHARED RESOURCE CONTROLLER

Due to the examination of the tasks required to be performed by the shared resource controller and the power already existing in a standard processing element, it is viewed that the best implementation of the shared resource controller would be in a standard PE. Cost wise the solution is the best due to the manufacture of a sixty-fifth identical unit instead of the design of a new piece of hardware. Of course, it is not performance optimized but nevertheless should give reasonably good performance due to the power existing in the PE. Much depends on the firmware used in the PE to execute the shared resource controller function. Since a firmware executive program will be vendor supplied with a suggested use scenario, the performance should be no problem.

The processing element, as a shared resource controller, will use only the Interprocessor Communications receive port for external communication. Existing plans will be to include the shared resource controller on the central communications segment along with the interfaces for the IOP, FCP, and PMS. One space per segment bus will be provided for the addition of more shared resource controllers in teh system. An additional quality that using a standard processing element gives will be the reliability of the system in the case of failure of the shared resource controller. Assuming the failure will not be within the BIU, (the BIU specifically decodes the SRC addresses portion of the SRC) it will be possible to switch new standard PE boards into the SRC mechanical space and continue the run.

6.1 Examination of Tasks

Upon examination of the shared resource controller function, the tasks have been divided into three major areas. One is concerned with communication of the SRC with the various PE's. The physical process of communication is handled the same way any IPC message is handled through the BIU within the PE. The ingest of several different requests at once is a deeper problem and depends directly on the priority that the firmware places on emptying the BIU's input buffers and storing the requests in an orderly manner within the PE. The other phase of communication deals with formatting "grant" messages in the SRC to send to the appropriate PE and the receipt of release signals from the PEs to allow another grant to be issued for that resource.

The second major task to be handled by the SRC is the choosing of the next highest priority process to which a grant must be issued once a release signal has been received. The process requests must be stored in an orderly manner with easy access to the microprogram which is executing the priority algorithm.

The third task for the SRC is the upkeep of the passage of pseudo-time representing the arbitration delay of the shared resource request. The method of upkeep of pseudo-time for any partiuclar resource is independent of the method for the other resources in the SRC domain. There are various methods by which the upkeep may be accomplished. The service time for the resource could be added and accumulated for each process awaiting that resource each time a higher priority process is being serviced. The differential of master pseudo-time between "request time" and "grant time" could represent the arbitration delay. The method chosen depends on the nature of the resource handled and pseudo-time synchronization accuracy desired. The independance of up-keep methods suggest the correct manner of division of labor among several SRC's that amy be in the system. The priority of pseudo-time upkeep method cannot be divided among the SRC's. A Clearer way of division is by resource. Any particular SRC would be responsible for one or more shared resources.

6.2 Firmware Use Scenario

For the purposes of ease of use and uniformity throughout the usage of the SRC, a suggested standard set of firmware methods will be highly desirable. The rest of this section will deal with one suggested structure of firmware design in the SRC. Of course, the vendor supplied PE executive and arbitration routines will completely specify and require the use of structured firmware. Even though the user may utilize the PE architecture to perform the SRC function in any manner that he chooses, there will be no guarantee of the ease of use and reliability of a nonstructured format.

6.2.1 Processing Element Communication

When a processing element needs to make a request to the shared resource controller, there are several parameters which must be passed to allow the SRC to perform its function. The message consists of the sending PE identification, the shared resource identification and some type of priority number associated with his request. This information is stored within the SRC. The grant message to the PE consists of the shared resource identification and the arbitration delay. In some cases it is necessary for the PE to send a release message indicating that it had seized the resource or completed action with that resource. This message resembles the request message except that a release message type takes the place of the request message type. In other cases a release message is not required. The SRC controller can make use of the bank of interval timers present in the PE to determine when the next grant message can be extended. The method used depends upon the type of arbitration that exists in the target machine. If arbitration is done on a cycle by cycle basis, the bank of interval timers will provide a regular way of spacing the grant messages. If a process is allowed to seize a resource for an undetermined period of time, the employment of a release message is necessary.

REQUEST MESSAGE

Request message type	PE Identification	Shared Resource Number	Priority

GRANT MESSAGE

SRC Message ID	Shared Resource ID	Arbitration Delay

RELEASE MESSAGE

Release Message Type	PE Identification	Shared Resource ID

6.2.2 Internal Firmware Structure

The internal firmware structure can be viewed logically as two control blocks. One is the resource control block (RCB) and the other is the process control block (PCB). The RCB contains all the information necessary about a particular shared resource. Particularly, it holds the entry point in microcode for the priority algorithm, the service time of the resource, the beginning address for the set of PCB's associated with the resource, and the busy/released status of the resource. The PCB exists to record a particular process that is awaiting a resource. Specifically it stores the reuqesting PE identification, the priority level of the request, the accumulated arbitration delay of the process, and the service/waiting status of the process. Both RCB's and PCB's are stored in internal memory. A possible block format follows:

RCB	PCB
µ- code entry address	PE ID Priority
resource service time	Arbitration Delay
PCB start address	Service/Waiting Status
No. of PCB's Busy/Release	

Upon receipt of a request, the SRC microcode examines the RCB to determine the status of the resource. After placing the request information into the PCB, the decision is made whether or not to handle the arbitration of this particular resource. If this resource is chosen either the current request or another process request already waiting may be issued the grant. When a grant is issued for a particular resource, the arbitration delay accumulated in the PCB is sent to the corresponding PE along with the SRC message ID and the shared resource ID. The remaining PCB's associated with the resource have to be individually updated by addition of the service time of the resource to the wait time accumulation field of the PCB. In order for a grant to be released by the SRC the busy/release field in the RCB must be in the released state. This field is set busy when a grant is issued and cleared by a release message sent by the PE. Of course when arbitration is done by cycle, the busy release field is not used. Grants are issued upon order by an interval timer.

7.0 PERFORMANCE MONITORING SUBSYSTEM (PMS)

The PMS will be discussed here in the form of three categories: features, implementation, and alternatives. The section outlining the PMS features will explain what functions the PMS will be capable of performing as well as what will be expected of the user and what the costs are in overhead. Implementation will outline generally what will be required of the vendor to implement PMS.

7.1 PMS Features

The key feature is a total integration of the PMS with the MMS hardware. Total integrations allows the user to debug, verify, and evaluate the emulated target system (TS).

7.1.1 Interactive Debug

Since there is a close relationship between the data that is collected for debug purposes and the data that is collected for the PMS, debug has been incorporated into the PMS. Within the scope of interactive debug the user has the following capabilities:

- o Single Step
- o Trace
- o Traps
- o Breakpoints
- o Examine and Change
- o Continue

7.1.2 Verification and Evaluation

As well as the debug capabilities the TS user has the following monitoring abilities:

- o Instruction Monitoring
- o I/O Monitoring

- o Memory Access Monitoring
- o Register Access Monitoring
- o Shared Resource Monitoring

Other user capabilities are listed below:

- o The user can dump the contents of all the registers, the PC, or any memory location.
- o The user can bump all memory addresses and data used during a particular instruction (this amounts to a trace during the execution of that particular instruction).
- o The user can indicate a point in the code or a pseudo-time at which a trap is inserted that allows a return of control to the user. Upon regaining control the user can choose to execute any of the PMS user functions.
- o The user can specify a dynamic (during the run) change in PMS data collection.

7.1.3 User Responsibilities

The main point to be made here is that the user's responsibilities are separated into the following two categories:

- o The microcode writer, who handles only the coding of the TS processor emulation.
- o The TS user, who is responsible for the utilization o[illegible]e microcode modules (the TS processor emula- [illegible]ons) and their interconnections and memory, [illegible] The TS user is also the writer (or user) [illegible] to be run on these TS processors. The[illegible] [illegible]ategories will usually be performed by by [illegible]ferent individuals.

7.1.3.1 The Microcode Writer

The microcode writer cannot be expected to be aware of what performance monitoring will be done in the final system. This assumption is the key to the following list of microcode writer responsibilities concerning PMS:

- o The microcoder will not be expected to write additional microcode for the sole purpose of performance monitoring.
- o The microcoder will document the code that is written in prespecified fashion. The documentation will be used later by the PMS processor.
- o The microcoder will design the microcode in a manner that is compatible with the hardware design of the PE.

7.1.3.2 The TS User

After configuration, the TS user will have to make the decision concerning what points should be monitored. This process of deciding will be done for each module in the TS. The TS user will have an interactive PMS command language that assists in the definition of monitoring points within the TS.

7.1.4 Overhead

Overhead caused in the TS by PMS is best defined as the time lost (emulation efficiency degradation) due to the performance monitoring. This time loss includes all categories of PMS that take up time such as deciding if PMS data is being collected, deciding if that particular piece of data is important, dumping the data, or analyzing it on-line. The following items are constraints put on the amount of overhead PMS can cause:

- o With no PMS data collection there is no overhead.
- o Checking to see if a particular item is of importance to the PMS causes overhead only when the item is a monitor point.

- When data is deemed important to the PMS, the user can tolerate a minimum amount of overhead necessary to dump the data. This actual data is the "payoff" of PMS and the overhead for its collection isn't that important.
- Overhead concerned with runtime analysis is dependent upon the analysis routine.

7.2 Implementation

In this section many references are made to Appendix A which is a complete, explanatory report on the PMS. Appendix A provides a detailed discussion of topics outlined in this section. The following topics are briefly presented in this section:

- Microcode structure
- Hardware Requirements
- Monitoring done outside the PE
- Overhead
- User Interaction

7.2.1 Microcode Structure

For a detailed description of how the microcode and PE hardware perform the decoding of macroinstructions see Appendix A, section 2.1.

7.2.2 Hardware Requirements

Additional internal memory (IM) is required in each PE for the implementation of the PMS. Each location in IM will have "flag bits" that indicate PMS collection as well as what additional data should be retrieved.

One flag bit will be checked by hardware to decide if

PMS is activated for that particular piece of data. If the bit is set (do collect for PMS) then an interrupt to PEXE will occur for the actual dumping of the event code or further checking of flag bits for possibly more collection.

The good point here is the lack of overhead caused by checking the one bit to determine if PMS is enabled. The PMS feature that states no overhead will occur if the PMS is not enabled is satisfied with this technique. For more explanation of the hardware requirements, see section 2.2 of Appendix A.

7.2.3 Monitoring Done Outside the PE

Monitoring of memory accesses and shared resource accesses are handled in the memory interface (MI) and the shared resource controller (SRC). Further explanation is in section 2.3 of Appendix A.

7.2.4 Overhead

There is no overhead associated with the "checking" for PMS enabled. When several monitoring points are enabled within the system but none of these points are accessed the system overhead looks as though PMS were disabled completely. Since performance data is often collected on a few specific events, the emulation efficiency is degraded very little and directly proportional to the number of times the events occurred. For a more complete discussion of "overhead" see section 2.4 of Appendix A.

7.2.5 User Interaction

There are two users of the MMS, the microcode writer who is responsible for the target processor emulations and the target system (TS) user who puts the microcode modules together to form an emulated multiprocessor system. Using the method presented here for implementation of the PMS the microcode writer must perform a bookkeeping task which implies very careful documentation of where opcode addresses, registers etc. are in IM. This documentation provides the PMS processor the necessary information to assist in monitor point enabling. The structured documentation also provides the microcoder a useful tool for debug and modification of the code. The TS user enables the points of interest with an interactive PMS command language.

The main points here are that the microcoder does not have to know what PMS points are of interest to any TS user and the microcode itself, does not have to be modified for PMS collection. Also, the TS user does not have to go into the microcode or the IM to enable the necessary PMS collection. There is more detail on the subject in Appendix A, section 2.5.

7.3 Summary

A complete summary of the PMS and how it satisfies the goals is in Appendix A, section 3.0.

All of the features of the PMS are satisfied using the method discussed in this report. All monitoring is handled in the IEU, the SRC, or the MI. All decisions concerning what data to dump and the actual dumping is handled in PEXE.

The microcode writer does not have to modify the microcode or be familiar with the MMS hardware.

The overhead required for the PMS is proportional to the amount of data being collected.

Overall, the PMS implemented as discussed here, requires little user interaction. Also, PMS only halts pseudo-time when data is being dumped. It is also a flexible implementation in that data collection can be changed and the user can stop, modify, and continue all during an applications run.

8.0 PRELIMINARY MECHANICAL DESIGN

A preliminary mechanical design of how the MMS can be mechanically configured as a 16 rack group will be presented in this section. Mechanically the MMS will be divided into two components:

1) The FCP minicomputer and its associated peripherals.
2) The MMS processing and control elements.

A floor plan for the MMS is given in figure 8.1.

8.1 The FCP

The FCP will be a stand alone mainframe minicomputer. The FCP will most likely be housed in as a 3 rack group.

8.2 The MMS

The analysis of the PE design yielded approximately 820 Dips/PE. A packaging analysis yielded that an average of 150 to 160 dips can be functionally placed on a large (9 inches by 15 inches) wire wrap card.

The wire wrap card chassis will hold a maximum of 13 wire wrap cards. Each MMS rack will in turn hold 3 wire wrap card chassis, 3 power supplies, and 2 fans.

8.2.1 The Master Segment

The master segment will be housed in one rack which will contain 2 distinct card chassis. The first card chassis type will contain the interfaces unique to the master segment. These interfaces will include the FCP interface (2 cards), the IOP interface (3 cards), the PMS interface (2 cards), and the TAC (1 card). The second chassis type will contain three functions that will be replicated on each PE segment. The Shared Resource Controller (6 cards), the Breadcast Controller (1 card), and the Bus Controller (1 card) will be the functions contained in this chassis. The third chassis will be a distribution box. The bus distribution box will connect a total of 11 elements to the master segment. These are:

- o A TAC
- o The PMS, IOP, and FCP interfaces
- o A Master BCC
- o An SRC
- o A BCU
- o Four ISBIU/BC Units

A mechanical configuration of the master segment is shown in figure 8.2.

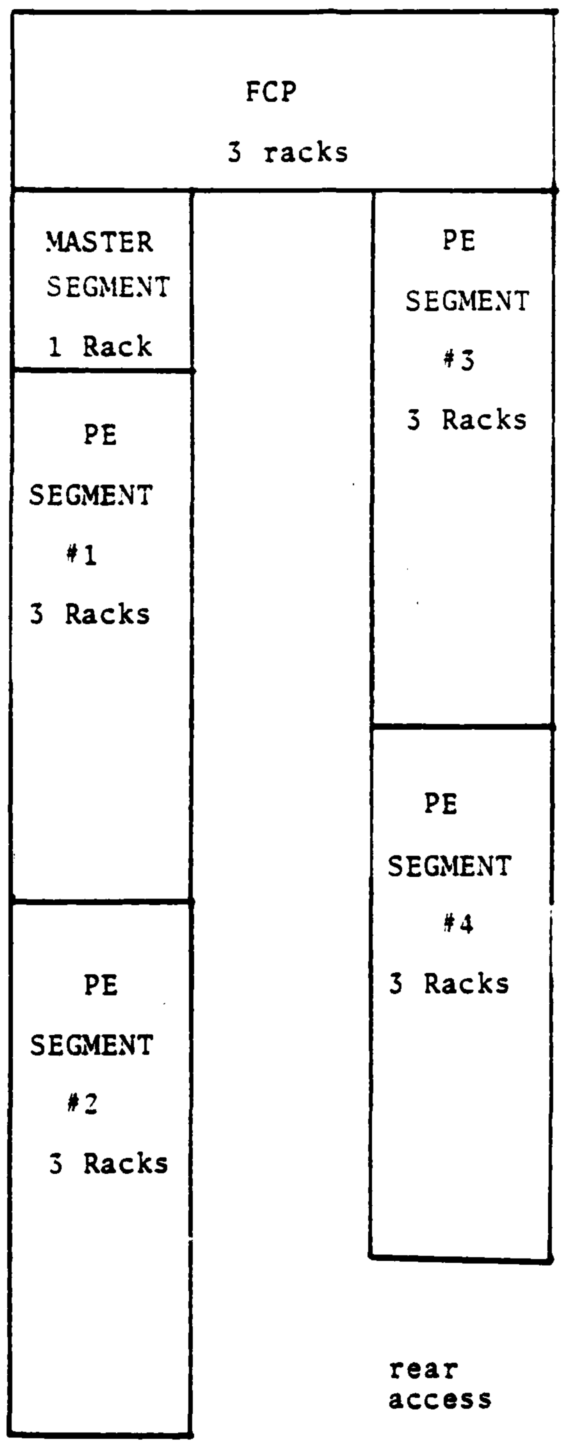
8.2.2 The PE Segments

The PE Segments will be 3 rack configurations. Two of the racks will contain 6 PE's each. The third rack will contain 4 PE's, a BCC, a ISBIU/BCU and a spare space for a future segment SRC.

Figure 8.2.2 provides a mechanical layout for the 3 racks of a PE segment.

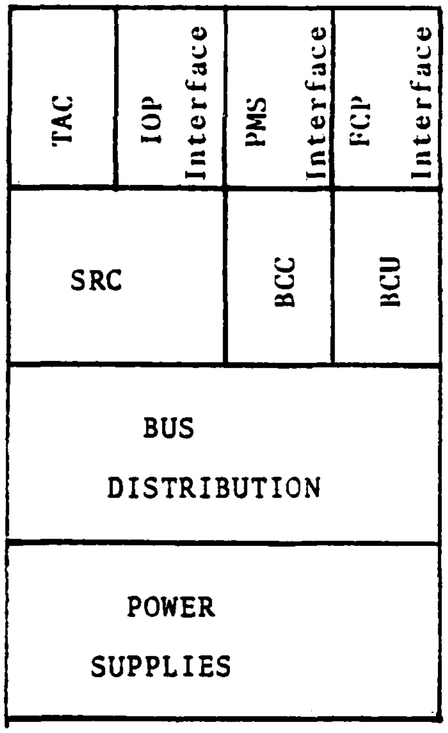
8.2.3 Power and Electrical Considerations

This hardware will require a computer room environment of 25^{o}C and will consume approximately 25 Kilo watts of AC power. The heat output will be in the order of 10,000 BTU's/hr..



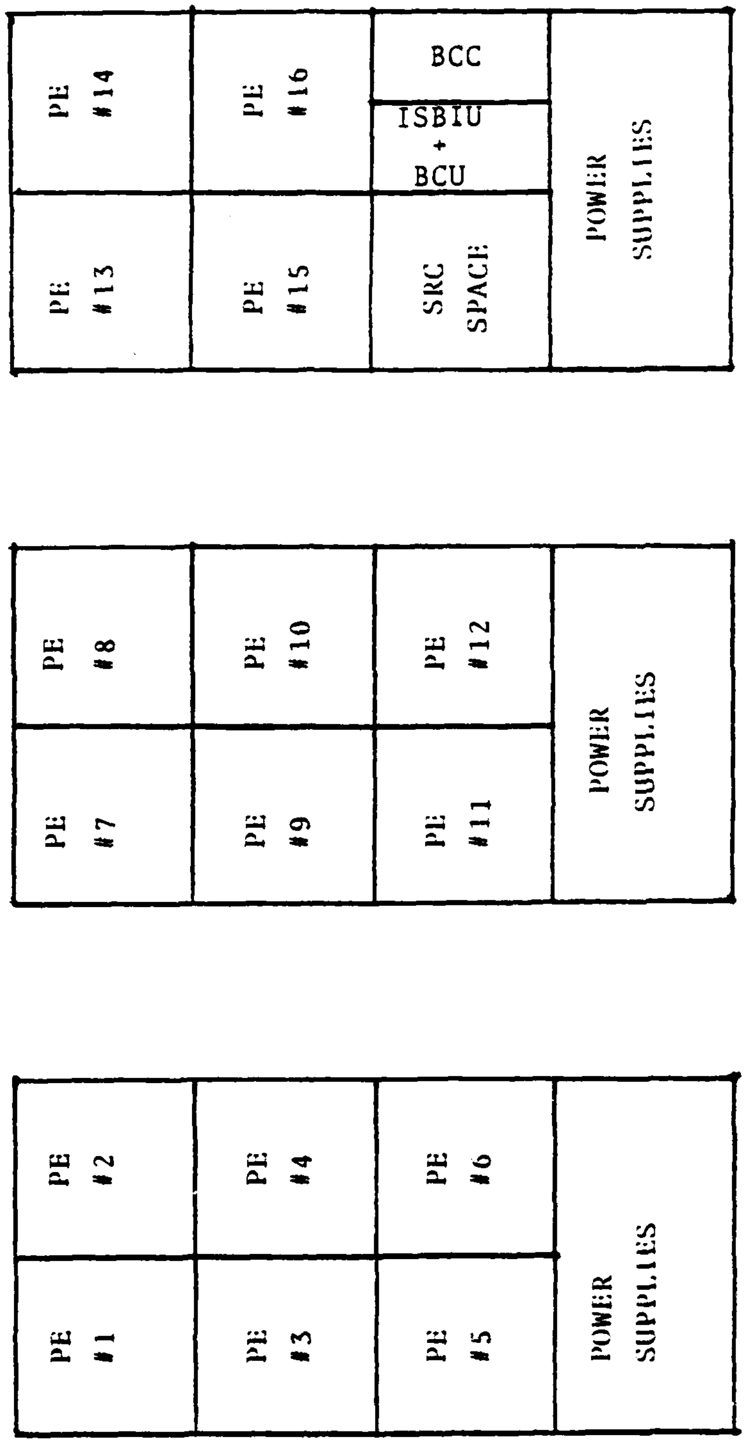
MMS FLOOR DIAGRAM

FIGURE 8.1



MASTER SEGMENT RACK

FIGURE 8.2.1



A PE SEGMENT
FIGURE 8.2.2